ДЕНИС БУТОРИН

bhv

MS Agent и Speech API в DELPHI

МЕТОДЫ, СВОЙСТВА И СОБЫТИЯ Microsoft Agent

ТЕХНОЛОГИИ СИНТЕЗА И РАСПОЗНАВАНИЯ РЕЧИ TEXT-TO-SPEECH И SPEECH RECOGNITION

РЕЧЕВЫЕ ИНТЕРФЕЙСЫ ВЫСОКОГО И НИЗКОГО УРОВНЕЙ

СОЗДАНИЕ СОБСТВЕННОГО АНИМИРОВАННОГО ПЕРСОНАЖА

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Денис Буторин

MS Agent и Speech API в DELPHI

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06
ББК 32.973.26-018.1Delphi
Б93

Буторин Д. Н.

Б93 MS Agent и Speech API в Delphi. — СПб.: БХВ-Петербург, 2005. — 448 с.: ил.

ISBN 5-94157-502-5

Рассмотрено программирование нестандартных пользовательских интерфейсов в среде Delphi с применением технологий Microsoft Agent и Microsoft Speech API для операционных систем Windows 98/2000/XP. Представлены способы внедрения анимированных персонажей в приложения Delphi и использования методов синтеза и распознавания речи с помощью функций Speech API. Подробно рассмотрены речевые интерфейсы высокого и низкого уровней. Описан процесс создания собственных анимированных персонажей и использования нестандартной текстовой выноски BalloonDialog.

На компакт-диске помещены все примеры, описанные в книге, необходимые компоненты и модули для создания приложений, а также программы автора, созданные с использованием описанных в книге технологий.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.1Delphi

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Галина Смирнова*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.11.04.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 36,12.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

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

ISBN 5-94157-502-5

© Буторин Д. Н., 2005
© Оформление, издательство "БХВ-Петербург", 2005

Содержание

Введение **1**

О чем и для кого эта книга 1
Структура и особенности книги 2
Программные и аппаратные требования 3

Глава 1. Технологии COM, OLE и ActiveX **5**

1.1. Технология Component Object Model 5
1.2. Базовые понятия 6
Объект 6
Интерфейс 7
Интерфейс *IUnknown* 8
Сервер 9
Библиотека COM 9
Другие элементы COM 10
1.3. Технология COM в среде Delphi 11
Объявление интерфейсов 11
Реализация интерфейсов 12
Другие способы реализации интерфейсов 16
Интерфейсы и класс *TComponent* 17
Создание подключаемых модулей *Plugins* с помощью интерфейсов 18
Первый COM-объект 22
Добавление второго интерфейса 26
1.4. Технология OLE Automation 31
Базовые понятия 31
Интерфейс *IDispatch* 32
Создаем OLE Automation Server 34
1.5. Технология ActiveX в среде Delphi 38
Краткие сведения 38
Создаем свой элемент ActiveX 39
Внедрение и использование компонента ActiveX 41
1.6. Резюме 45

Глава 2. Технология MS Agent в среде Delphi 47

2.1. Что такое MS Agent? 47
Microsoft Agent 47
Установка компонентов и импорт ActiveX 49
Лицензия на использование и распространение 54
Использование программного обеспечения 54
Авторские права: использование изображений 55
Компоненты пользовательского интерфейса 56
2.2. Советы по применению MS Agent в приложениях 61
Основные рекомендации 62
Рекомендации социального характера 63
Рекомендации по синтезу речи 64
Советы по распознаванию речи 64
Выводы 65
2.3. Объектная модель Microsoft Agent Object 66
Объект *Agent Control* 67
Объект *Request* 68
2.4. Начало программирования персонажей 68
Прямое обращение к Agent Server 70
Раннее и позднее связывание с элементом управления Agent Control 77
Программирование MS Agent с ActiveX 85
2.5. Свойства объекта *Request* 87
2.6. Свойства, события, методы и объекты элемента управления Agent Control 94
Методы 94
Свойства 96
События 100
Объекты 107
2.7. Объект *Character* элемента управления Agent Control 109
Свойства персонажа как элемента управления 109
"Личные" свойства персонажей 110
Основные свойства и методы объекта *Character* 115
Доступ к персонажам через Интернет 123
Методы остановки запросов 133
Галерея Default Character Properties — своими руками 141
Получаем список анимаций 143
Раскрывающееся меню и объект *Commands* 147
Персонаж и справочная служба 154
"Воздушный шарик" персонажа (Balloon) 156
Синхронизация действий персонажей 165
Много приложений + много персонажей 168

2.8. Речевой вывод и ввод 179
Базовые понятия....179
Объект *AudioOutput* 180
Учимся говорить 183
Коррекция речи 190
Объект *SpeechInput*....195
Распознавание команд, или еще раз об объекте *Commands* 195
2.9. MS Agent на web-страницах 206
2.10. Резюме 208

Глава 3. Технология Speech API в среде Delphi 211

3.1. Введение в речевые технологии....212
О чем это мы? 212
Что напишем, то и слышим 215
Текст превращается в речь 216
Как это делается? 217
Компьютер управляется голосом....218
Долой клавиатуру?....220
Предъявите голос 221
О некоторых речевых программах....222
3.2. Основы технологии Speech API....224
Необходимые компоненты и их установка 224
Speech-технологии....231
Компоненты SAPI 232
3.3. Интерфейс синтеза речи Voice Text API 236
Необходимые интерфейсы и объекты....236
Первая программа чтения текстов 241
Совершенствуем первую "читалку" 244
Изменение атрибутов синтеза речи....247
Диалоги....250
Доступ к объекту *VoiceText* через OLE Automation....252
Работа с объектом *VoiceText* через ActiveX 256
3.4. Интерфейс DirectTextToSpeech API....259
Основные COM-объекты 260
Приложение с использованием интерфейса DirectTextToSpeech API 262
Атрибуты речи 273
Теги 283
Диалоги....290
Обработка уведомлений (notification sink)....290
Синтез речи в аудиофайл 294
Полное приложение с интерфейсом DirectTextToSpeech API 297

3.5. Интерфейс распознавания речи Voice Command API 298
COM-объекты интерфейса Voice Command API 298
Приложение с объектом *VoiceCommand* 299
Списки имен 305
Объект *VoiceCommand* и главное меню 306
Объект *VoiceCommand* в позднем связывании 307
Объект *VoiceCommand* и ActiveX 308
3.6. Интерфейс распознавания речи Voice Dictation API 310
Основные понятия. Объекты Voice Dictation API 311
Приложение с объектом *VoiceDictation* 312
Объект *VoiceDictation* и ActiveX 315
3.7. Интерфейс синтеза речи DirectSpeechRecognition API 316
Основные понятия. COM-объекты 316
Создаем приложение с интерфейсом DirectSpeechRecognition API 318
Параметры речевых движков 327
Атрибуты распознавания речи 330
Подготовка грамматик 332
Элемент управления DirectSR 335
3.8. Резюме 336

Глава 4. "Примочки" к MS Agent 339

4.1. Апплет Microsoft Speech Control Panel 339
4.2. Редактор персонажей Microsoft Agent Character Editor 343
Практические советы по анимированию 343
Основные понятия и принципы 345
Начинаем создавать свой персонаж 346
Подготовка изображений 351
Создаем анимации 354
Привязка анимаций к состояниям персонажа 359
Анимации возвращения (Return Animation) 360
Переходы (Branching) 361
Многокартинные кадры 363
Звуковые эффекты 366
Анимации рта 366
4.3. Лингвистический редактор Microsoft Linguistic Information Sound Editing Tool 369
Начало работы 369
Создание лингвистических LWV-файлов 370
Создание и редактирование фонем и слов 373
4.4. Компонент BalloonDialog от фирмы SommyTech 377
Начало работы 379
Минимальное приложение 380

Основные свойства и методы BalloonDialog 388
Метод *MsgBalloon* 395
Метод *InputBalloon* 398
Метод *TipBalloon* 399
Метод *Suggest* 402
Объект *FormBalloon* и метод *ShowFormBalloon* 403
4.5. Совмещаем технологии MS Agent и Speech API 408
4.6. Резюме 411

Заключение 413

Приложения 415

Приложение 1. Ссылки на сайты и файлы в Интернете 417

Ссылки на сайты 417
Ссылки на файлы 418

Приложение 2. Описание компакт-диска 421

Список литературы 424

Предметный указатель 425

Введение

О чем и для кого эта книга

Предметом книги является подробное изложение принципов программирования нестандартных пользовательских интерфейсов в среде Delphi с применением таких технологий, как Microsoft® Agent Character v2.0 (MS Agent) и Microsoft® Speech API 4.0 (SAPI 4.0) в операционных системах Windows 98, Windows 2000 и Windows XP. В ней рассказывается о способах внедрения анимированных персонажей MS Agent в приложения Delphi и о методах синтеза речи с помощью функций SpeechAPI. Книга содержит описание отличительных особенностей программирования в разных версиях Delphi. Кроме того, здесь описаны другие дополнительные компоненты и модули, используемые в технологии MS Agent. На прилагаемом компакт-диске помещены все описанные в книге примеры, а также компоненты и модули, необходимые для создания приложений с использованием технологий анимированных виртуальных персонажей MS Agent и синтеза речи средствами SpeechAPI.

Технология MS Agent создана корпорацией Microsoft и впервые была применена в приложениях MS Office. Наверняка все знают забавных персонажей MS Word — Скрепку, Мурку и т. п. Однако автор уверен, что не только Microsoft нравятся эти помощники. Некоторые программисты, особенно начинающие, хотели бы научиться использовать технологию анимированных персонажей в своих собственных приложениях, а MS Agent может пригодиться как в больших офисных приложениях в качестве интеллектуального помощника, так и в игровых программах, различных напоминалках и т. п.

Технология синтеза и распознавания речи Text-To-Speech и Speech Recognition также разработана компанией Microsoft®. В этой области имеются большие достижения, и, очевидно, развитие ее будет продолжаться. Не исключено, что сторонние разработчики пожелают применить подобные технологии в своих приложениях.

Существует немало англоязычных материалов по этой теме в применении к программированию на языках C/C++ и Visual Basic, а вот подробной документации для программистов Delphi недостаточно. Данная книга претендует на то, чтобы заполнить создавшийся пробел. Автор считает, что язык Delphi (начиная с версии 7 он называется так официально) ничем не хуже C/C++, и что, пользуясь им, можно осуществить почти все замыслы. Естественно, для каждого языка программирования определен конкретный круг решае-

мых задач, но это не значит, что для каких-либо из них допустимо отсутствие документации по узким темам.

Предлагаемая вашему вниманию книга представляет собой наиболее полное руководство по программированию в среде MS Agent, а также первую документацию на русском языке по использованию технологии SAPI 4.0.

Книга рассчитана на широкий круг читателей — от начинающих программистов Delphi до достаточно подготовленных. Тем не менее, автор надеется, что читатель уже знаком с объектно-ориентированным программированием (ООП) и базовыми понятиями Delphi.

Структура и особенности книги

Книга состоит из четырех глав, построеных таким образом, чтобы их можно было читать независимо друг от друга. Тем не менее, в книге встречаются примеры программных кодов, которые связаны между собой (например, в *гл. 4*, в описании совмещения технологии MS Agent и SpeechAPI). Поэтому, если вы никогда раньше не работали с той или иной технологией, вам следует читать эти главы по порядку.

Глава 1 "Технологии COM, OLE и ActiveX" посвящена основам, необходимым для понимания технологий COM (Component Object Model) и ActiveX, а также началам программирования компонентов COM в Delphi. Она включает минимальный набор базовых знаний по COM-технологии и не заменяет собой какой-либо другой подробной и детальной документации. Если вы новичок в Delphi и (или) никогда не слышали о технологиях автоматизации, то вам необходимо прочитать эту главу, чтобы далее свободно понимать механизмы работы MS Agent и SpeechAPI. Если же вы представляете себе устройство COM или считаете, что владеете данной темой, то можете перейти к следующим главам.

В главе 2 "Технология MS Agent в среде Delphi" вам предстоит первое знакомство с персонажем MS Agent. Здесь рассказывается о его установке в операционной системе и среде программирования. Детально описываются практически все его интерфейсы, свойства, события и методы.

Глава 3 "Технология Speech API в среде Delphi" познакомит читателей с технологией SAPI4, с ее основными интерфейсами прикладного программирования: Voice Text API, Voice Command API, Voice Dictation API, Direct Text-To-Speech API и Direct Speech Recognition API. Здесь также рассказывается об основных компонентах, которые необходимы при использовании речевых технологий. В этой главе мы научимся писать приложения, синтезирующие речь, записывающие ее в аудиофайл. Мы рассмотрим в ней также вопросы распознавания речи с помощью SAPI и займемся созданием приложений, распознающих отдельные голосовые команды и поддерживающих диктовку.

Глава 4 "«Примочки» к MS Agent" представляет собой описание некоторых наиболее интересных дополнительных к MS Agent модулей и компонентов (так называемых "примочек"). Это готовые программные модули, приложения или ActiveX-компоненты, расширяющие функциональность персонажа. В этой главе мы научимся создавать свои собственные анимированные персонажи, подготавливать специальные лингвистические файлы для улучшения воспроизведения волновых файлов приложением MS Agent. Здесь же мы познакомимся с наиболее популярным ActiveX-компонентом для персонажа-агента BalloonDialog© от фирмы SommyTech®.

На прилагаемом компакт-диске записаны все примеры, обсуждаемые в книге с учетом различных версий Delphi. Те примеры, которые хорошо переносятся на Delphi 6 и Delphi 7, написаны в 5-ой версии среды. Программный код, который требует хотя бы незначительных поправок, дан в вариантах для других версий среды программирования. Кроме того, компакт-диск содержит все необходимые модули для написания приложений с использованием MS Agent и Speech API.

Программные и аппаратные требования

Для работы с MS Agent рекомендуется иметь операционную систему Windows начиная с версии 95 и выше и приложение Internet Explorer версии v3.02 и выше. Необходимы: процессор модификации не ниже Pentium 100 МГц, объем оперативной памяти не менее 16 Мбайт, а также Windows-совместимая звуковая карта. Из дополнительных модулей для пользователей Windows 9*x* необходимы MS Agent Control v2.0 и один или несколько файлов персонажей. Ссылки вы найдете в *приложении 1.*

Для работы с синтезом и распознаванием речи *(см. гл. 3)* приведенные минимальные программные требования немного повышаются. Хотя это, в принципе, неважно, т. к. программное и аппаратное обеспечение компьютеров даже у рядовых пользователей на сегодняшний момент вполне удовлетворяет этим требованиям. Из дополнительных модулей необходимо установить набор функций Speech API v4.0 и один или несколько речевых синтезаторов (движков). Так как мы попытаемся распознавать речь, то ко всем установленным модулям надо добавить движки распознавания речи.

При изучении материала четвертой главы для создания собственных анимированных существ потребуется редактор персонажей Microsoft Agent Character Editor, а при рассмотрении вопросов создания лингвистических файлов — лингвистический редактор Microsoft Agent Linguistic Information Sound Editing Tool. Ну и, естественно, вам понадобится компонент BalloonDialog 6.5 для изучения примеров, иллюстрирующих работу "шаровоздушной" выноски.

Глава 1

Технологии COM, OLE и ActiveX

В данной главе будут рассмотрены основы COM-модели и ее спецификации. Мы познакомимся с объявлением интерфейсов в Delphi и научимся ими пользоваться. Затем напишем пример с размещением COM-объекта в DLL-библиотеке и в завершение познакомимся с технологией ActiveX и OLE Automation, создав простейший ActiveX-компонент и OLE Automation Server.

Начинающему программисту в Delphi необходимо прочитать эту главу для дальнейшего понимания всех конструкций и технологии решений в последующих главах. Однако данный материал не является заменой специальной литературы по COM-технологии. Он всего лишь позволит познакомиться с реализацией COM-технологии в теории и на практике.

1.1. Технология Component Object Model

С тех пор как появилось программирование, перед разработчиками больших проектов стоит задача обеспечения связи между отдельными программными модулями. Для решения этой нетривиальной задачи было разработано множество средств. Ко многим из них мы уже давно привыкли, например, к буферу обмена, разделяемым файлам, технологии динамического обмена данными (DDE — Dynamic Data Exchange). Технология OLE (*Linking and Embedding* — связывание и внедрение объектов) была разработана для обеспечения обмена данными и предоставления служб. Она решает общую задачу предоставления программными модулями друг другу собственных функций.

Существует множество других разработок, но во всех них в основе лежит базовая технология *Component Object Model* (COM) — многокомпонентная модель объектов или модель объектных компонентов. Это одна из главных технологий, на которых основывается Windows. С помощью COM один программный модуль предоставляет свои функции, а другие части программного обеспечения получают доступ к ним. Эти части могут находиться

как в одном процессе, так и в разных, и даже на различных компьютерах. Причем совершенно неважно, на каком из языков программирования реализованы эти части — при соблюдении спецификации COM они будут работоспособны в любой ситуации.

На основе COM разработаны такие технологии, как ActiveX, ActiveForm, Microsoft Transaction Server, DirectX, Shell и др.

COM является весьма успешной технологией, потому что позволяет строить мощные распределенные системы с минимумом усилий. Очевидные ее достоинства — в способности приложения COM развиваться с течением времени, удобстве и гибкости при модернизации существующих приложений.

Среда Delphi предоставляет разработчикам возможность создавать полноценные приложения, использующие технологию COM.

1.2. Базовые понятия

Рассмотрим базовые понятия, знание которых необходимо для работы с технологией COM.

Объект

Ключевым понятием модели COM является *объект*. Это естественно, т. к. со времени появления объектно-ориентированного программирования (ООП) все сложные системы разбиваются на классы, экземпляры которых обладают собственными свойствами и методами. COM-компонент должен удовлетворять некоторым требованиям, приведенным ниже.

- Он должен скрывать используемый язык программирования.
- Объекты должны распространяться в двоичной форме, без необходимости компиляции и компоновки, т. е. готовыми к использованию.
- Должна быть предусмотрена возможность модернизировать компоненты, не затрагивая уже существующих пользователей. Новые версии компонента должны работать как со старыми, так и с новыми клиентами.
- Компоненты должны перемещаться по сети. Необходимо, чтобы компонент и использующий его клиент могли выполняться внутри одного процесса, в разных процессах и на разных машинах.

Итак, COM-приложение использует COM-объекты для предоставления своих функций клиентам. Стало быть, приложение должно иметь как минимум один COM-объект для того, чтобы считаться приложением COM.

Как и любой объект, COM-объект соответствует почти всем требованиям ООП. Он является экземпляром какого-либо класса, т. е. представляет собой переменную объектного типа. COM-объект имеет набор методов и

свойств и удовлетворяет трем основным требованиям: *инкапсуляции* (скрытие данных от прямого доступа), *полиморфизму* (свойство вызова методов в соответствии с классом фактически переданного параметра), а также некоторым элементам *наследования*.

Интерфейс

Что делает COM-объект объектом COM, что заставляет его отличаться от остальных объектов? Как бы мы поступили, разрешая противоречие об обмене данными между приложениями на месте разработчиков COM?

Чтобы установить обмен данными между двумя разнородными системами (между клиентом и сервером), необходимо создать нечто общее, т. е. заранее "объяснить" компонентам, как они будут общаться. Это осуществляется с помощью одного из центральных элементов модели COM, называемого *интерфейсом*.

Интерфейс в модели COM — это средство, которое позволяет клиенту правильно обратиться к объекту COM, а объекту позволяет ответить клиенту так, чтобы он (клиент) его (сервер) понял. В идеологии COM каждый объект может поддерживать один или более интерфейсов, а каждый интерфейс — включать в себя несколько методов объекта для обеспечения доступа к его свойствам и выполнения определенных действий.

Что такое интерфейс?

Вообще говоря, в самом широком смысле для интерфейсов любого типа существует неофициальное определение: интерфейс — это *то*, посредствам чего *что-то* соединяется с *чем-то*. Например, интерфейсом между нами и клавиатурой компьютера являются наши руки. Они позволяют нам — послать сообщение серверу (клавиатуре) о нажатии клавиши, а клавиатуре — вернуть клиенту (нам) ответ о том, что клавиша была нажата. Оригинальный пример, неправда ли? Я специально его придумал, чтобы читатель, незнакомый с COM, быстро запомнил это очень важное понятие: "интерфейс".

Строгое определение интерфейса (в применении к COM) в литературе может иметь следующую формулировку.

Интерфейс — это описание на языке программирования виртуальной таблицы адресов функций, поддерживаемых объектом. Подобные виртуальные таблицы аналогичны виртуальным таблицам методов (VMT — virtual method table) классов Delphi.

Каждый интерфейс имеет два *атрибута*. Первый атрибут — собственно название интерфейса, составленное в соответствии с правилами языка программирования. Имя должно начинаться с буквы "I" — так принято, так же как в Delphi название любого класса должно начинаться с буквы "T".

Второй атрибут представляет собой *глобальный уникальный идентификатор* (Globally Unique IDenitfier, GUID), который задается уникальным сочетанием символов. Для его генерации используются такие сильные алгоритмы, что однажды сгенерированное сочетание никогда не повторится в будущем ни на одном компьютере мира.

Существует особенность в характере наследования у объектов COM. В случае наследования интерфейса код реализации должен представить потомок, а наследование реализации подразумевает полное включение программного кода родителя в код потомка. Однако реализация при первом варианте наследования никуда не исчезает, объекты используют механизм включения и при необходимости потомок вызывает метод родителя.

Если интерфейс представляет собой своеобразную палочку-выручалочку для получения доступа к методам объекта, то как же тогда к ним обратиться? Для этого необходимо получить указатель на соответствующий интерфейс, после чего клиент имеет право использовать службы объекта, вызывая его методы как методы обычного объекта.

Следует еще пояснить некоторые детали. Поскольку объект может имеет несколько интерфейсов, то при получении интерфейса для каждого из них будет получен собственный указатель. Иначе говоря, при вызове процедур получения интерфейса вы получаете не один указатель для всех интерфейсов объекта, а только тот, который был запрошен. Это удобно, потому что гораздо рациональнее однообразные функции объединять в отдельные интерфейсы.

Возникает закономерный вопрос — как быть, если клиент не знает, какие интерфейсы имеются у объекта? Для получения их перечня нужно использовать базовый интерфейс `IUnknown`, который есть у любого объекта COM. К рассмотрению этого интерфейса мы сейчас и перейдем.

Интерфейс *IUnknown*

`IUnknown` является базовым интерфейсом COM-объектов. Через этот интерфейс можно получить все остальные интерфейсы, которые поддерживает объект. В нем присутствуют всего три метода, но они играют самую важную роль в функционировании объекта:

- `QueryInterface`
- `AddRef`
- `Release`

Метод `QueryInterface` возвращает указатель на интерфейс объекта по его идентификатору. Если передан идентификатор несуществующего интерфейса, то метод возвратит `Null`.

Кроме предоставления независимого от языка программирования доступа к методам объектов, COM реализует автоматическое управление памятью для COM-объектов, основанное на идее подсчета ссылок на объект. Объект существует, пока его использует хотя бы один клиент. Поэтому любой клиент после создания объекта должен увеличить счетчик ссылок, а после завершения его использования уменьшить счетчик на единицу. Когда счетчик достигнет нулевого значения, COM-объект автоматически будет удален из памяти. Именно для этого предназначены метод `AddRef`, увеличивающий счетчик на единицу, и метод `Release`, уменьшающий его.

Стоит заметить, что, пользуясь функцией `QueryInterface` для получения указателя на интерфейс в среде Delphi, метод запускает процедуру увеличения счетчика ссылок. Вызвать его вручную может потребоваться в том случае, если один клиент попытается передать другому указатель на интерфейс объекта. Тогда без вызова `AddRef` счетчик будет хранить неверные сведения о количестве использующих его клиентов. То же справедливо и для метода `Release`. При выходе переменной, ссылающейся на интерфейс, за область видимости (либо при присвоении ей другого значения) компилятор Delphi генерирует код для вызова метода `Release`, информируя реализацию COM-объекта о том, что ссылка на нее больше не нужна. Поэтому нет надобности постоянно вызывать этот метод, за исключением особых случаев.

Сервер

Объект работает в составе *сервера COM*. Сервер может быть как динамической библиотекой, так и исполняемым файлом. Объект имеет свои собственные свойства и методы или пользуется данными и службами сервера.

COM-серверы бывают трех типов:

- *внутренний сервер* — обычно это динамические библиотеки, подключаемые к приложению. Функционируют в одном адресном пространстве, поэтому называются еще *in-process server*;
- *локальный сервер* — для такого сервера создается отдельный процесс, работающий на одном компьютере с приложением-клиентом;
- *удаленный сервер* — работает в своем процессе на другом компьютере по отношению к клиенту.

Библиотека COM

Итак, мы разобрались с тем, как получить указатель на интерфейс — для этого используется метод `QueryInterface`. Но что же делать, если клиенту требуется получить интерфейс объекта, который еще не использовался и, следовательно, может быть и не создан. В этом случае клиенту необходимо обратиться к *библиотеке COM*. Она обеспечивает выполнение базовых

функций и интерфейсов в операционной системе. К ней обращаются посредством специальных функций, имена которых согласно спецификации начинаются с приставки `Co`.

При установке COM-приложения в системный реестр записываются данные обо всех реализуемых им объектах.

- `CLSID` (`Class Identifier`) — идентификатор класса, однозначно идентифицирует класс объекта в системе;
- тип сервера объекта (внутренний, локальный, удаленный):
  - для внутренних и локальных серверов записывается полное имя динамической библиотеки или исполняемого файла;
  - для удаленных — полный сетевой адрес.

Поэтому для получения указателя на требуемый класс и интерфейс необходимо вызвать метод `CoCreateInstance`, передав в качестве параметров CLSID нужного класса, IID интерфейса (Interface IDentifier) и тип требуемого сервера.

Возвращение указателя происходит по следующей схеме.

1. Библиотека через диспетчер управления службами обращается к системному реестру.
2. Находит информацию о сервере по идентификатору класса CLSID.
3. Запускает сервер.
4. Сервер создает экземпляр класса.
5. Объект возвращает библиотеке указатель на запрошенный интерфейс.
6. Библиотека COM передает указатель клиенту.

Другие элементы COM

В данном разделе мы не рассмотрели некоторые другие базовые элементы COM, однако они не менее важны.

Для запуска экземпляра класса служит специальный объект — *фабрика класса*. С его помощью можно создать один или несколько экземпляров некоторого класса. То есть для каждого класса существует своя фабрика класса.

Разработчик COM-объектов для пользователей создает специальную информацию, описывающую типы объектов, чтобы документировать интерфейсы. Вся информация составляется на языке IDL и объединяется в специальной *библиотеке типов*.

Все вышесказанное может показаться сложным, и тем не менее, это так только на первый взгляд. В следующем разделе мы перейдем непосредственно к программированию и "пощупаем" все своими руками.

1.3. Технология COM в среде Delphi

Сейчас мы познакомимся со спецификой реализации COM-приложений в Delphi. На первом этапе изучим способы объявления интерфейсов в собственных приложениях, а затем рассмотрим детали их реализации. После этого узнаем, как обратиться к главному приложению из дополнительных модулей, таких как DLL, с помощью интерфейсов. Наконец, создадим полноценное COM-приложение с несколькими интерфейсами и методами.

Объявление интерфейсов

В Delphi существует специальное ключевое слово, расширяющее набор `Object Pascal` для поддержки интерфейсов. Этим ключевым словом является слово `interface`.

```
IFirstInterface = interface
  //описание интерфейса
end;
```

Мы уже знаем, что каждый интерфейс имеет имя и уникальный идентификатор GUID. С именем все более или менее понятно: оно придумывается разработчиком и начинается с буквы `I`. В данном случае интерфейс носит имя `IFirstInterface`.

Однако что делать с GUID — глобальным идентификатором? Для глобального идентификатора в Delphi имеется специальный тип `TGUID`:

```
TGUID = packed record
  D1: LongWord;
  D2: Word;
  D3: Word;
  D4: array[0..7] of Byte;
end;
```

В прикладном интерфейсе программирования Windows (WinAPI) существует специальная функция для генерации нового идентификатора — `CoCreateGUID`. Она описана в модуле `ActiveX.pas`.

Для преобразования идентификатора в строку и обратно используются соответствующие функции из модуля ComObj.pas:

```
function GUIDToString(const ClassID: TGUID): string;
function StringToGUID(const S: string): TGUID;
```

В Delphi имеется и более простой способ генерации нового идентификатора. Для вставки нового идентификатора в нужное место необходимо установить туда курсор и нажать комбинацию клавиш <Ctrl>+<Shift>+<G>. Действуя

по этому способу, в описании нашего нового интерфейса поставьте курсор на следующую строку и сгенерируйте идентификатор. Получится результат, подобный приведенному ниже:

```
IFirstInterface = interface
['{34381141-4B31-11D8-8903-0020ED19BE94}']
  //Описания методов интерфейса
end;
```

Теперь можно описывать требуемые методы, которые должен включать описываемый интерфейс. Например, объявим ничего не возвращающий метод `Hello` без параметров:

```
IFirstInterface = interface
['{34381141-4B31-11D8-8903-0020ED19BE94}']
   procedure Hello();
end;
```

Теперь можно приступать к реализации методов.

Реализация интерфейсов

Реализацией интерфейса в Delphi всегда выступает класс. Поэтому в объявлении класса необходимо указать, какие интерфейсы он реализует.

> ***О реализации***
>
> Возвращаюсь к примеру неофициального определения интерфейсов. Может показаться странным, что нельзя просто написать реализацию в секции `implementation`. Однако в этом случае интерфейс не отличался бы от обычного класса. В ситуации, когда мы являемся клиентом, наша клавиатура — сервером, а наши руки — интерфейсом, реализация интерфейса не находится у нас в руках. Метод нажатия клавиши реализует клавиатура! В ней реализовано то, как именно должны замыкаться контакты, как должен подаваться сигнал в компьютер.

Рассмотрим пример объявления класса, реализующий интерфейс.

```
type
  IFirstInterface = interface
  ['{34381141-4B31-11D8-8903-0020ED19BE94}']
     procedure Hello();
  end;

  TFirstClass = class(TInterfacedObject, IFirstInterface)
    procedure Hello();
  end;
```

Подобное объявление не свидетельствует о наличии множественного наследования в Delphi в том виде, как оно реализовано в C++. Это говорит о том, что в данном классе реализованы все указанные интерфейсы.

Класс должен иметь методы, точно соответствующие по именам и спискам параметров всем методам, объявленным в его заголовке интерфейсов. То есть необходимо написать реализацию всех методов интерфейсов, указанных в объявлении класса, причем имена этих методов должны совпадать с именами методов, указанных в описании интерфейса, а также должны совпадать списки их параметров.

Новый класс является наследником от `TInterfacedObject`, рекомендуется именно его использовать для реализации своих собственных интерфейсов.

Полный код примера объявления и реализации интерфейса расположен в каталоге Chapter1\01_FirstInterface\ на прилагаемом компакт-диске.

Теперь рассмотрим более подробный пример (см. листинг 1.1). Опишем некоторый интерфейс и реализуем его в классе `TFirstClass`, а затем запустим какой-нибудь из его методов. Запуск будет производиться щелчком мыши на кнопке `Button1`.

Листинг 1.1. Демонстрация работы интерфейса (проект из каталога 02_Simple)

```
type
  IFirstInterface = interface
  ['{34381141-4B31-11D8-8903-0020ED19BE94}']
     procedure Hello(); stdcall;
  end;

  TFirstClass = class(TInterfacedObject, IFirstInterface)
    procedure Hello(); stdcall;
    destructor Destroy; override;
  end;

  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.DFM}
//Реализации интерфейса
procedure TFirstClass.Hello(); stdcall;
begin
 ShowMessage('Hello!')
end;

destructor TFirstClass.Destroy;
begin
 ShowMessage('Запустился деструктор!');
end;

//Обработка щелчка мыши на кнопке:
procedure TForm1.Button1Click(Sender: TObject);
var
 localVar: IFirstInterface; //Локальная переменная
begin
  localVar:=TFirstClass.Create;//Приводим класс к интерфейсу
  localVar.Hello();//Вызываем метод
end;
```

Приведенный в листинге 1.1 код может показаться странным в том плане, что мы нигде не увеличиваем и не уменьшаем счетчик ссылок, а также не освобождаем память из-под объекта `localVar`. Однако приведение класса к интерфейсу неявно увеличивает счетчик ссылок на единицу. Кроме того, при выполнении программы появится сообщение о том, что был вызван деструктор. Это происходит потому, что при выходе переменной, ссылающейся на интерфейс, за область видимости (либо при присвоении ей другого значения) компилятор Delphi генерирует код для вызова метода `_Release`, информируя реализацию о том, что ссылка на нее больше не нужна. Таким образом, после выполнения первой строки кода счетчик увеличится на единицу, а при выходе из процедуры уменьшится на единицу, и объект будет удален из памяти.

Рассмотрим теперь код (см. листинг 1.2), который вызовет появление ошибки вследствие неправильной работы с классом, реализующим интерфейс (пример из подкаталога 03_ErrorCode).

Листинг 1.2. Демонстрация ошибки (пример из каталога 03_ErrorCode)

```
procedure TForm1.Button1Click(Sender: TObject);
var
 VInt: IFirstInterface;
 VCls: TFirstClass;
begin
 VCls := TFirstClass.Create;
 VInt := VCls;//Передали указатель другой переменной
 VInt.Hello;//Отработала процедура
 VCls.Free;//Удаляем реализацию интерфейса
 //После выполнения этой процедуры будет ошибка
end;
```

При выходе из процедуры возникнет ошибка, т. к. при передаче указателя интерфейсу счетчик увеличится на единицу. После удаления реализации останется указатель на интерфейс, и, соответственно, при выходе переменной за область видимости неявно будет вызван метод `_Release`. Тут и произойдет исключение недействительной операции с указателем, т. к. реализация будет уже удалена.

Поэтому никогда не вызывайте деструктор класса явно — объект будет удален, как только отпадет необходимость в его последней ссылке. Если в данном случае вам необходимо удалить реализацию немедленно, нужно просто присвоить указателю значение `nil`, как показано в листинге 1.3.

Листинг 1.3. Демонстрация правильного кода

```
var
 VInt: IFirstInterface;
 VCls: TFirstClass;
begin
 VCls := TFirstClass.Create;
 VInt := VCls;//Передали указатель другой переменной
 VInt.Hello;//Отработала процедура
 VInt:=nil;//После этого неявно будет вызван IUnknown._Release
           //и уничтожена реализация, т. к. отпадет необходимость
           //в последней ссылке
end;
```

Другие способы реализации интерфейсов

Нередко встречаются ситуации, когда класс должен реализовывать несколько интерфейсов. В этом нет ничего сложного, т. к. необходимо всего лишь, чтобы методы класса совпадали по именам и спискам параметров с методами, объявленными в его заголовке интерфейсов. Но как поступить в случае, когда в нескольких различных интерфейсах присутствуют одноименные методы? Например, в случае, показанном в приведенном ниже фрагменте кода:

```
IFirstInterface = interface
['{34381141-4B31-11D8-8903-0020ED19BE94}']
   procedure Hello();
end;

ISecondInterface = interface
['{34381142-4B31-11D8-8903-0020ED19BE94}']
   procedure Hello();
end;
```

При описании реализации методов интерфейсов в каком-либо классе возникнет переобъявление методов. Выйти из положения можно следующим образом (см. также проект в каталоге 04_TwoInterface):

```
TFirstClass = class(TInterfacedObject, IFirstInterface, ISecondInterface)
     procedure Hello1;
     procedure Hello2;
     procedure IFirstInterface.Hello = Hello1;
     procedure ISecondInterface.Hello = Hello2;
end;
```

Конфликт разрешается явным указанием на то, какой метод служит реализацией соответствующего метода интерфейса.

Если реализация методов должна быть идентичной, то обычно проблему решают, назначая один и тот же метод класса реализацией нескольких методов интерфейсов. То есть поступают следующим образом:

```
TFirstClass = class(TInterfacedObject, IFirstInterface, ISecondInterface)
     procedure Hello;
     procedure IFirstInterface.Hello = Hello;
     procedure ISecondInterface.Hello = Hello;
end;
```

Демонстрационный проект находится в подкаталоге 05_TwoInterface2.

Интерфейсы и класс *TComponent*

В базовом классе `TComponent` имеется полный набор методов, позволяющих реализовать интерфейс `IUnknown`, хотя сам класс данный интерфейс не реализует. Это позволяет наследникам `TComponent` реализовывать интерфейсы, не заботясь о реализации `IUnknown`.

Методы `TComponent._AddRef` и `TComponent._Release` на этапе выполнения программы не реализуют механизм подсчета ссылок, т. е. в отношении классов-наследников `TComponent`, реализующих интерфейсы, не действует автоматическое управление памятью. Это позволяет запрашивать у них интерфейсы, не опасаясь, что объект будет удален из памяти при выходе переменной за область видимости.

В качестве примера (см. листинг 1.4) рассмотрим аналогичный приведенному в листинге 1.2 код, который работает корректно по сравнению с кодом из проекта 03_ErrorCode.

Листинг 1.4. Демонстрация корректного кода (проект каталога 06_GoodCode)

```
type
  IMyInterface = interface
  ['{DE9ADBC0-4CD7-11D8-8903-0020ED19BE94}']
    procedure Hello();
  end;

  TIntClass = class(TComponent, IMyInterface)
    procedure Hello();
  end;

  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}
```

```
procedure TIntClass.Hello();
begin
  ShowMessage('Hello!');
end;

procedure TForm1.Button1Click(Sender: TObject);
var
 VInt: IMyInterface;
 VCls: TIntClass;
 i: Integer;
begin
 VCls:=TIntClass.Create(Self); //Обратите внимание на параметр Self
 VInt:=VCls;//Передали указатель другой переменной
 VCls.Hello();//Отработала процедура
 i:=VCls._Release;
 ShowMessage(IntToStr(i));//Здесь мы получим сообщение "-1"
 VInt.Hello();//А вот в этом месте уже НЕ будет ошибки!
end;
```

Благодаря классу `TComponent` в методе `Create` появился параметр `AOwner: TComponent`, т. е. "хозяин". Поэтому `VCls` может быть удален из памяти методом `Free` вручную, хотя он также очистит память при уничтожении формы, т. к. теперь она является хозяином данного объекта.

Создание подключаемых модулей *Plugins* с помощью интерфейсов

Те объекты, которые мы создавали в предыдущих разделах, еще нельзя назвать полноценными COM-объектами, т. к. они не входят в состав COM-сервера, информация о них не записывается в реестр и они не выполняют базовой функции — осуществления обмена данными между приложениями (вспомните, с чего все началось).

Тем не менее, проблемы, которые будут рассмотрены в данном разделе, невозможно решить с помощью обычных классов ООП без применения интерфейсов.

Представим себе такую ситуацию. Существует приложение типа записной книжки, в которой реализованы функции чтения, записи и т. п. Однако вы желаете обеспечить импорт в приложение данных из других форматов.

Соответственно, необходимо реализовать идею подключаемых модулей — plugins ("плагинов"). Модуль знает, как прочитать данные в определенном формате и записывает их в базу данных или в файл основной программы. В этой идее есть один нюанс. Что же получается — нам необходимо предоставлять прямой доступ к данным главной программы? (В нашем случае к базе данных или файлу). Однако вам вряд ли понравится, если ваш коллега напишет подключаемый модуль, который однажды удалит какие-нибудь важные данные. Возможно, во второй раз вы им и не воспользуетесь, да только информация уже будет удалена.

Очевидно, что в подобном случае главная программа должна предоставлять интерфейсы конечным разработчикам, закрывая для доступа детали реализации внутреннего механизма работы программы.

Надеюсь, что многие читатели сталкивались с созданием *динамических библиотек*. Обычно динамическая библиотека (DLL-библиотека) предоставляет функции вызывающему приложению. Теперь же мы попробуем построить модуль и главное приложение так, чтобы и из вспомогательного модуля (!) можно было вызвать некоторые функции главной программы.

Для удобства реализации мы упростим заданную ситуацию и создадим пример, в котором возможно из DLL-библиотеки изменить значение некоторой переменной главной программы, причем реализация определенного метода интерфейса будет контролировать допустимые значения (в данном случае — не допускать ввод значений меньших −100).

Начнем с того, что описание интерфейса (см. листинг 1.5) должно находиться в отдельном модуле. Его мы будем распространять среди конечных разработчиков.

Листинг 1.5. Модуль описания интерфейса

```
unit InterfaceUnit;

interface

uses
  Classes;

type
  IPluginInterface = interface
  ['{34381143-4B31-11D8-8903-0020ED19BE94}']
     procedure Hello();
```

```
    function GetLocalVar(): integer;
    procedure SetLocalVar(sInt: integer);
  end;

implementation

end.
```

Этот модуль мы подключим к модулю основной формы главного приложения (см. листинг 1.6), причем укажем там, что класс главной формы реализует требуемый интерфейс (или интерфейсы) и опишем реализацию методов интерфейса, как делали раньше.

Листинг 1.6. Листинг модуля главного приложения (секции `interface` и `implementation` значительно упрощены)

```
unit Unit1;

interface

uses
  ..., InterfaceUnit;

type
  TForm1 = class(TForm, IPluginInterface)
  ...
  public
    { Public declarations }
    procedure Hello();
    function GetLocalVar(): integer;
    procedure SetLocalVar(sInt: integer);
  end;

var
  Form1: TForm1;
  ...

implementation
```

```
var
 TestInt: integer;

//Реализация интерфейса
procedure TForm1.Hello();
begin
 ShowMessage('Привет из главного приложения!')
end;

function TForm1.GetLocalVar(): integer;
begin
 Result:=TestInt;
end;

procedure TForm1.SetLocalVar(sInt: integer);
begin
 if sInt >= -100 then
   TestInt:=sInt
 else
   ShowMessage('Нельзя присвоить значение меньше -100!')
end;

...

end;
```

Затем опишем загрузку библиотеки и вызов функций обычным способом (подробнее см. коды основной программы проекта каталога 07_Plugins).

Теперь перейдем к DLL. Строим ее как обычную библиотеку, подключаем в раздел `uses` модуль, в котором описаны необходимые интерфейсы, и создаем процедуры и функции для инициализации плагина и его выгрузки. Рассмотрим этот этап подробнее.

В процедуре инициализации модуля необходимо передавать из главного приложения объект `Application` для того, чтобы получить доступ к главной форме программы. Естественно, основное приложение должно передавать значение своего объекта `Application`. Для удобства будем преобразовывать тип `TApplication` к типу `Integer`.

После проделанных действий мы можем обратиться к главной форме приложения и запросить у нее указатель на нужный интерфейс:

```
var
  Plg: IPluginInterface;

procedure InitPlugin(App: integer); StdCall;
begin
  ...
  Application := TApplication(App);
  Application.MainForm.GetInterface(IPluginInterface, Plg);
end;
```

В данном примере указатель на интерфейс получается посредством вызова функции:

```
function GetInterface(const IID: TGUID; out Obj): Boolean;
```

Так как `MainForm` является свойством класса `TApplication`, то метод `QueryInterface` из секции `protected` класса `TCustomForm` для него не поддерживается. Однако ничто не мешает воспользоваться методом `GetInterface`, реализованным еще в классе `TObject` в секции `published`.

На этом можно поставить точку, т. к. через указатель на интерфейс, хранящийся в переменной `Plg`, можно получить доступ ко всем методам, которые поддерживает данный интерфейс.

Запустите основную программу. Попытайтесь изменить значение переменной `TestInt` и получить его в форме, вызванной из DLL. Затем, наоборот, измените значение переменной в дополнительном модуле и посмотрите, чему будет оно равно в главном. Значения всегда будут совпадать, т. к. мы работаем с одной переменной. Обратите внимание на то, что невозможно задать число меньшее −100. Контроль ввода происходит в реализации главной программы, т. е. от конечного разработчика ничего не зависит. Тем самым мы скрыли детали реализации нашего приложения.

Подобным образом можно реализовать запись необходимых данных в базу данных и файл. Необходимо всего лишь, чтобы в главном приложении были соответствующим образом реализованы методы интерфейсов.

Первый COM-объект

Настало время создать наш собственный COM-объект. Включим его в состав динамической библиотеки.

Пусть наш сервер называется `FirstServer`, объект — `FirstCOM`. Объект будет включать один-единственный интерфейс — `IFirstCOM` с методом `GetSqr`.

Этот метод будет возвращать квадрат переданного числового значения. Конечно, для нахождения квадрата числа совершенно необязательно пользоваться моделью COM, однако для нас лучше показать общую схему на простом примере.

Для начала необходимо создать ActiveX Library — будущий сервер. В меню **File | New** (Файл | Новый) на вкладке **ActiveX** следует выбрать именно этот тип приложения, т. к. в данном случае Delphi сгенерирует необходимый код для вызова служебных функций. Перед вами откроется заготовка библиотеки — это и будет наш сервер. Теперь добавим к нему COM-объект. Снова на той же вкладке **ActiveX** следует выбрать пункт **COM Object** (COM-объект). Появится диалоговое окно примерно такого вида, как показано на рис. 1.1.

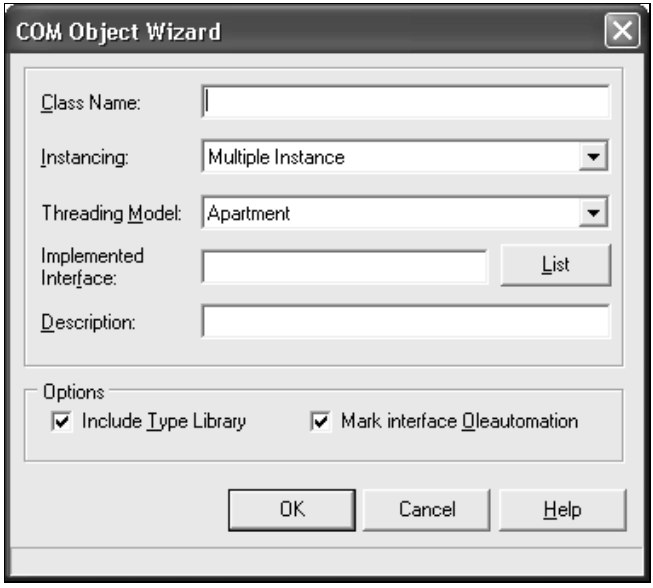


Рис. 1.1. Диалоговое окно мастера создания COM-объектов

Верхнее поле редактирование **Class Name** (Имя класса) определяет имя нового класса.

Раскрывающийся список **Instancing** (Типы объекта) определяет способ создания объекта:

- **Internal** (Внутренний объект) — объект используется в процессе приложения. В этом случае никакие данные не записываются в реестр, т. е. объект не будет доступен внешним процессам;

- **Single Instance** (Единственный) — при обращении к единственному экземпляру сервера нескольких клиентов создается необходимое число объектов;
- **Multiple Instance** (Множественный) — при обращении нескольких клиентов для каждого создается собственный экземпляр сервера объекта.

Раскрывающийся список **Threading Model** (Модель потоков) определяет способ взаимодействия объекта и клиентов:

- **Single** (Однопоточная модель) — сервер может обслуживать вызовы только последовательно один за другим;
- **Apartment** (Раздельная) — вызов объекта осуществляется только в одном потоке, который создается самим объектом, в сервере в таком режиме одновременно могут работать несколько объектов, каждый в своем потоке. При этом данные, распределенные между экземплярами объектов (например, глобальные переменные), должны быть защищены объектами синхронизации потоков;
- **Free** (Свободная) — вызов объекта может осуществляться через любой созданный им поток. В этом случае COM-объект должен защищать даже свои собственные данные от одновременного доступа из нескольких потоков;
- **Both** (Раздельная и свободная вместе) — объект может работать с клиентами в модели Apartment и Free.

После внесения необходимых значений и щелчка на кнопке **OK** к проекту будет добавлен новый модуль с исходным кодом нового класса, а также модуль с вариантом библиотеки типов на Object Pascal. Он будет назван так же, как и проект, с окончанием `_TLB` в названии и расширением pas. Для работы с библиотекой типов в Delphi существует специальный редактор. Он предоставляет довольно полный набор инструментов для автоматизации создания объектов и интерфейсов (рис 1.2). Редактор появится на экране сразу после добавления объекта.

Создайте интерфейс `IFirstCOM`, щелкнув на кнопке **New Interface** (Новый интерфейс), а затем метод `GetSqr`, щелкнув на кнопке **New Method** (Новый метод). Выделите метод, после появления новых вкладок перейдите на страницу **Parameters** (Параметры). Здесь необходимо добавить один входной целочисленный параметр, а также указать тип возвращаемого значения (целочисленный). Обновите исходный код модуля кнопкой **Refresh** (Обновить).

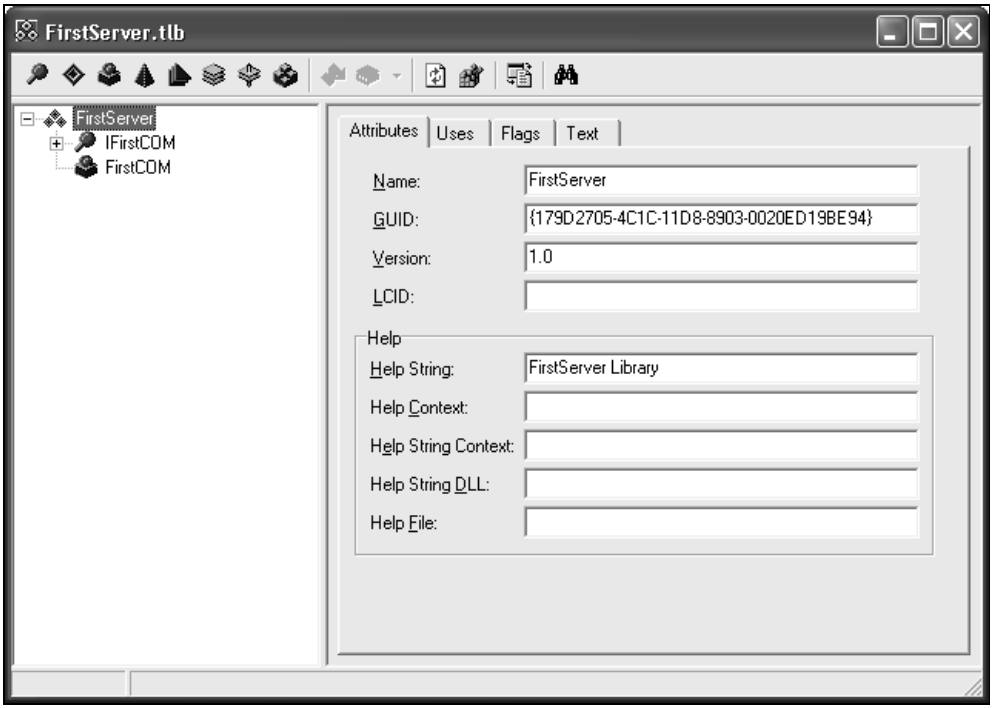


Рис. 1.2. Редактор библиотеки типов

В модуле описания класса `TFirstClass` в секции `implementation` найдите шаблон реализации метода `GetSqr` и впишите в него код возвращения квадрата переданного значения:

```
function TFirstCOM.GetSqr(aValues: SYSINT): SYSINT;
begin
  Result:=aValues*aValues;
end;
```

Сохраните все модули и откомпилируйте проект COM-приложения. После этого самое время зарегистрировать его в системе. Для этого выполните команду **Run | Register ActiveX Server** (Запуск | Зарегистрировать ActiveX Server), после чего должно появиться сообщение об успешной регистрации сервера в системе. При этом в системном реестре будет создан ключ вида:

```
HKEY_CLASSES_ROOT\CLSID\{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxx}
```

Перейдем к созданию клиентского приложения. Первым делом следует подключить необходимые модули в разделе uses — ActiveX.pas и библиотеку типов от нашего COM-объекта. Если вы назвали сервер FirstServer, то модуль библиотеки типов будет называться FirstServer_TLB.pas.

В обработке события создания главной формы приложения-клиента следует поместить код для обращения к библиотеке COM и получения указателя на интерфейс, т. к. клиент еще не имеет указатель на нужный объект и интерфейс:

```
var
  fIFirstCOM: IFirstCOM;

implementation

procedure TForm1.FormCreate(Sender: TObject);
begin
  CoCreateInstance(CLASS_FirstCOM, Nil, CLSCTX_ALL,
     IID_IFirstCOM, fIFirstCOM);
end;
```

В параметрах указан `CLSID` — идентификатор класса, типа сервера (третий параметр), идентификатор интерфейса, а также последний выходной параметр показывают, в какую переменную нужно поместить указатель. Идентификаторы класса и интерфейса представляют собой типизированные константы типа `TGUID`, объявленные в нашей библиотеке типов.

Ничего не стоит теперь вызвать необходимый метод интерфейса:

```
var
 i: integer;
begin
    ...
    i:=fIFirstCOM.GetSqr(i);
    ...
end;
```

Пример данного проекта находится в каталоге 08_COMObject.

Добавление второго интерфейса

Чтобы продемонстрировать идею получения одних интерфейсов COM-объекта через другие, создадим новое COM-приложение, аналогичное предыдущему (08_COMObject), под названием `SecondServer`. Добавим в него объект с именем `SecondCOM`. По умолчанию в нем будут созданы один интерфейс и один `CoClass`.

Разместим в приложении форму и создадим два интерфейса `IServerForm` и `ISizeForm`. Первый будет реализовывать методы для обеспечения показа

(ShowForm) и скрытия формы (HideForm), второй — методы сброса размеров окна (ResetForm) и максимизацию окна (MaximizeForm). Причем метод сброса размеров усложним добавлением входных параметров положения окна.

Итак, первый интерфейс переименуем в IServerForm, соответственно CoClass назовем ServerForm. Добавим два метода к интерфейсу с помощью кнопки **New Method** (Новый метод). Не будем пока описывать реализацию, а займемся созданием формы. Форма создается и присоединяется так, как обычно делается в других приложениях Delphi. После того как форма будет готова, необходимо удалить из ее модуля переменную типа класса созданной формы (см. листинг 1.7). Она располагается в секции interface непосредственно перед секцией implementation. При этом необходимо добавить объект класса формы в модуль, в котором описан класс, реализующий организованные интерфейсы. Естественно, в секцию uses этого модуля нужно включить модуль с описанием формы.

Перемещение объекта необходимо для того, чтобы из модуля реализации интерфейсов была доступна внедренная форма.

Листинг 1.7. Модуль формы FormCOM.pas (каталог 09_ProbaCOMForm\Server)

```
unit FormCOM;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

type
  TfrCOM = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

{Заметьте, на привычном месте отсутствует описание объекта класса нашей
формы. Объект должен быть описан в модуле с реализацией интерфейсов}
implementation
{$R *.DFM}
```

```
procedure TfrCOM.Button1Click(Sender: TObject);
begin
 Close
end;

end.
```

После проделанных действий можно добавлять второй интерфейс. Нажмите кнопку **New Interface** (Новый интерфейс), переименуйте его в `ISizeForm`. На вкладке **Attributes** (Атрибуты) в раскрывающемся списке **Parent Interface** (Родительский интерфейс) требуется выбрать предка, т. е. `IUnknown`. Добавьте два новых метода: `ResetForm` и `MaximizeForm`. Затем необходимо присоединить новый интерфейс к классу `ServerForm`. Для этого выделите класс в иерархическом списке, на вкладке **Implements** (Реализация) щелкните на списке правой кнопкой мыши и в появившемся меню выберите команду **Insert Interface** (Вставить интерфейс). В открывшемся диалоговом окне отметьте в списке доступных интерфейсов недавно созданный интерфейс. После обновления кода название нового интерфейса добавится в заголовке описания класса, а также появятся заготовки реализаций объявленных методов.

Теперь можно приступить к описанию реализации методов интерфейсов. После их описания модуль с исходным кодом класса примет вид, показанный в листинге 1.8.

Листинг 1.8. Модуль с исходным кодом класса (из проекта 09_ProbaCOMForm\Server)

```
unit UnitObj;
interface
uses
  Windows, ActiveX, Classes, ComObj, SecondCOM_TLB, StdVcl, FormCOM,
Forms;

type
  TServerForm = class(TTypedComObject, IServerForm, ISizeForm)
  protected
    {Declare IServerForm methods here}
    function HideForm: HResult; stdcall;
    function ShowForm: HResult; stdcall;
    function MaximizeForm: HResult; stdcall;
    function ResetForm(aX, aY: Integer): HResult; stdcall;
  end;
```

```
var
 frCOM: TForm;

implementation

uses ComServ;

function TServerForm.HideForm: HResult;
begin
 try
  frCOM.Hide;
  frCOM.free;
  Result:=S_OK;{Если удалось выгрузить форму, возвращаем положительный
результат}
 except
  Result:=GetLastError; {Иначе возвращаем код ошибки}
 end;
end;

function TServerForm.ShowForm: HResult;
begin
 try
  frCOM:=TfrCOM.Create(nil);
  frCOM.Show;
  Result:=S_OK;
 except
  Result:=GetLastError;
 end;
end;

function TServerForm.MaximizeForm: HResult;
begin
 try
  if frCOM<>nil then
    frCOM.WindowState:=wsMaximized;
  Result:=S_OK;
 except
  Result:=GetLastError;
 end;
end;
```

```
function TServerForm.ResetForm(aX, aY: Integer): HResult;
begin
 try
  if frCOM<>nil then
   begin
    frCOM.WindowState:=wsNormal;
    frCOM.Top:=aX;
    frCOM.Left:=aY;
    frCOM.Width:=300;
    frCOM.Height:=300;
   end;
  Result:=S_OK;
 except
  Result:=GetLastError;
 end;
end;

initialization
  TTypedComObjectFactory.Create(ComServer, TServerForm, Class_ServerForm,
    ciSingleInstance, tmApartment);
end.
```

Приступим к созданию приложения-клиента. Приложение будет во всем подобно клиенту из предыдущего проекта (см. проект 08_COMObject на компакт-диске). Не забудьте подключить библиотеку типов от COM-сервера (в нашем случае — SecondCOM_TLB.pas) и ActiveX.pas.

При обработке события создания главной формы приложения-клиента, через функцию `CoCreateInstance` библиотеки COM получим указатель на первый интерфейс (не имеет значения на какой — `IServerForm` или `ISizeForm`), например, на `IServerForm`, а затем через функцию `QueryInterface` получим указатель на интерфейс `ISizeForm`.

После получения указателя мы сможем воспользоваться методами, которые реализуют данные интерфейсы (см. листинг 1.9).

Листинг 1.9. Обработка события создания формы клиента в проекте из каталога 09_ProbaCOMForm\Client

```
var
  Form1: TForm1;
```

```
  fVsForm: IServerForm;
  fSzForm: ISizeForm;

implementation

procedure TForm1.FormCreate(Sender: TObject);
begin
  CoCreateInstance(CLASS_ServerForm, Nil, CLSCTX_ALL,
     IID_IServerForm, fVsForm);
  fVsForm.QueryInterface(IID_ISizeForm, fSzForm);
end;
```

В первый раз пользуемся функцией `CoCreateInstance`, т. к. отсутствует указатель на какой-либо интерфейс, т. е. COM-объект может быть и не создан еще вовсе, и поэтому обращаемся к библиотеке COM. Второй интерфейс запрашиваем через `QueryInterface`, потому что не имеет смысла обращаться опять к COM-библиотеке, раз указатель на один интерфейс COM-объекта получен — функция `QueryInterface` отработает быстрее, т. к. принадлежит интерфейсу `IUnknown`.

1.4. Технология OLE Automation

Поняв суть технологии COM, нетрудно заметить некоторое противоречие, возникающее при ее использовании. Так как стандарт COM основан на едином для всех поддерживающих его языков формате таблицы, описывающей ссылки на методы объекта, реализующего интерфейс, то можно сказать, что все это будет доступно из компилирующих языков. Однако как быть с интерпретаторами вроде VBScript? Для поддержки этих языков была разработана технология OLE Automation, позволяющая приложениям делать свою функциональность доступной для гораздо большего числа клиентов. Automation (автоматизация), основывается на COM, и является ее подмножеством.

Базовые понятия

Введем некоторые основные понятия.

Данные, которые используются совместно, называются *OLE-объектом*; приложения, содержащие OLE-объекты, называются *OLE-контейнерами*, а приложения, способные содержать свои данные в OLE-контейнерах, называются *OLE-серверами* или *серверами автоматизации*.

Для того чтобы COM-объект мог называться OLE-объектом, он должен удовлетворять следующим требованиям.

- Интерфейс, реализуемый COM-сервером, должен наследоваться от интерфейса `IDispatch`
- Должны использоваться типы данных из числа поддерживаемых OLE Automation. Возможна поддержка пользовательских типов данных, для чего необходимо реализовать интерфейс `IRecordInfo`.
- Все методы должны быть процедурами или функциями, возвращающими значение типа `HRESULT`.
- Все методы должны иметь соглашение о вызовах `safecall`.

Кроме того, для работы с OLE важно понимание двух подходов к задаче реализации доступа к COM-объектам: раннего и позднего связывания. Создание COM-объектов в предыдущих примерах данного раздела можно назвать ранним связыванием *(early binding)*, т. к. в основе этого подхода лежит то правило, что во время компиляции кода известен адрес метода объекта. Имеется в виду, что мы пользовались библиотекой типов и указывали интерфейс и метод, которые необходимо использовать, напрямую. Подобный подход можно применять для OLE-объектов в компилируемых языках. Однако другая идеология доступа — позднее связывание *(late binding)* — подразумевает, что вызов метода возможен на этапе выполнения программы, в обход прямого доступа, если на момент компиляции его невозможно узнать. Об этом мы поговорим ниже.

Надо сказать, что хотя позднее связывание считается медленным способом доступа, на практике это трудно почувствовать. При размещении сервера в отдельном исполняемом файле и, соответственно, при нахождении клиента и сервера в разных потоках затраты времени на раннее и позднее связывание будут практически равны. Это связано с механизмами реализации обоих принципов связывания. Однако если объект построен по технологии In-Process-Server (внутренний сервер) при совместимой с клиентом потоковой модели, т. е. сервер находится в одном процессе с клиентом, разница в затратах времени, естественно, будет ощутимой. Но нас это обстоятельство не должно пугать, т. к. создание COM-объекта ради использования его в том же процессе бесперспективно. Вспомните, к чему мы стремились — мы хотели обеспечить связь между приложениями и реализовать идею предоставления функций одних приложений другим.

Интерфейс *IDispatch*

Мы знаем, что каждый OLE-объект должен наследовать свой интерфейс от интерфейса `IDispatch`, следовательно необходимо выяснить, чем же отличается обычный интерфейс `IUnknown` от `IDispatch`. Я не буду, в целях эконо-

мии места, приводить описание этого интерфейса (вы сами можете найти его в справочной системе Delphi), а отмечу лишь отличия.

Во-первых, необходимо сказать, что `IDispatch` унаследован от `IUnknown` и имеет ряд дополнительных методов, которые и позволяют осуществлять позднее связывание. Рассмотрим эти методы поподробнее.

Новый метод `Invoke` позволяет вызвать метод, поддерживаемый объектом, через его диспетчерский идентификатор (Dispath ID) `DispID`. Метод `GetIdsOfNames` позволяет получить `DispID` любого метода через его имя.

Некоторые особо нетерпеливые читатели спросят: "Что за позднее связывание? Понять бы, чем оно отличается от раннего!" Что ж — привожу пример из подкаталога 10_LateBinding (см. листинг 1.10), реализующий создание документа MS Word из среды Delphi.

Листинг 1.10. Демонстрация позднего связывания

```
Uses
 ..., ComObj;
 ...
var
 MSWord: Variant;
begin
 MsWord := CreateOleObject('Word.Application');

 MSWord.Documents.Add;
 MSWord.ActiveWindow.Visible:=false;

 MSWord.Selection.Font.Size := Label1.Font.Size;
 MSWord.Selection.Font.Name := Label1.Font.Name;
 MSWord.Selection.Font.Color := Label1.Font.Color;
 MSWord.Selection.Font.Bold:=(fsBold in Label1.Font.Style);
 MSWord.Selection.Font.Italic:=(fsItalic in Label1.Font.Style);
 MSWord.Selection.Font.Underline:=(fsUnderline in Label1.Font.Style);
 MSWord.Selection.ParagraphFormat.Alignment:=ComboBox1.ItemIndex;
 MSWord.Selection.TypeText(Edit1.Text);

 MSWord.ActiveWindow.Visible:=true;
end;
```

На первый взгляд пример выглядит странно. Во-первых, переменная `MSWord` не является объектом какого-либо класса и, соответственно, не имеет никаких методов и свойств. С другой стороны, код успешно компилируется, как будто ничего особенного в нем нет. Ну, и уж совсем необычным кажется вызов приложения Word по его обычному (информативному) имени. Однако ничего странного в этом примере вы не найдете, если разберетесь в том, что за нас "додумывает" Delphi.

В самом начале данного кода мы ассоциируем переменную `MSWord` с приложением Microsoft Word с помощью функции `CreateOleObject` из модуля `ComObj`. При компиляции вызов метода автоматизации имеет вид вызова метода `Invoke`, а во время выполнения метод `Invoke` вызовет указанный метод автоматизации. То есть при вызове метода автоматизации через тип Delphi `Variant` (также можно использовать тип `OleVariant`) транслятор Delphi вызовет метод `GetIDsOfNames` для получения идентификатора `DispID` по имени необходимого метода автоматизации, а затем уже будет выполняться собственно метод автоматизации функцией `Invoke` через `DispID`. Все просто!

Вы спросите: "При чем тут `Invoke`, если переменная имеет тип `Variant`?" А при том, что функция `CreateOleObject` возвращает главный интерфейс объекта, унаследованный от типа `IDispatch`. Так как мы не знаем заранее, какой именно это тип, то используем тип `Variant`.

Все хорошо, но нам нужно знать названия методов, поддерживаемых объектом. Их можно выяснить по созданной разработчиками соответствующей документации и SDK (Software Development Kit). Что же касается приложения Word и других приложений MS Office, отмечу, что любые их команды можно изучать с помощью макросов. Вы записываете в макрос необходимые действия и затем просматриваете автоматически созданный код, в котором видно, какие методы и свойства используются для достижения результата. Код макроса можно перенести (отредактировав его в соответствии с синтаксисом Delphi или любого другого языка) в ваше собственное приложение.

Создаем OLE Automation Server

Думаю, что после создания собственного небольшого сервера автоматизации некоторые неясности в понимании технологии OLE исчезнут сами собой. Займемся же этим.

Для простоты создаваемый OLE-объект будет иметь один интерфейс (`ISimple`) с одним методом (`NextInt`), который будет принимать в качестве параметра целое число и возвращать следующее за ним по порядку (т. е. на единицу большее) целое число.

Работу следует начать с создания библиотеки ActiveX Library (точно так, как было сделано в примерах по созданию COM-объектов). Затем надо сформировать в данной библиотеке объект Automation Object (соответствующий

пункт находится в списке на вкладке **ActiveX** диалогового окна **New Items** (Новый пункт), вызываемого командой **Files | New** (Файл | Новый).

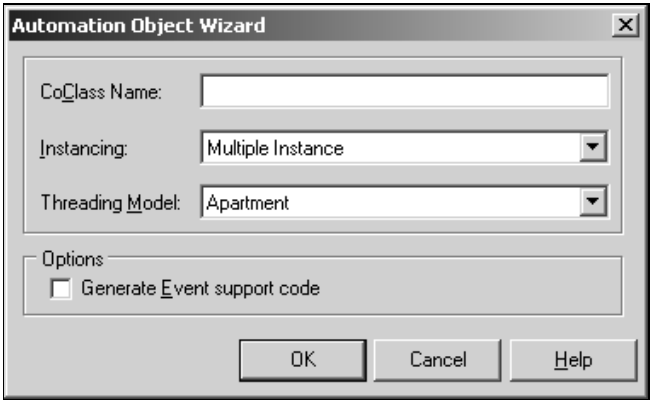


Рис. 1.3. Диалоговое окно мастера создания объекта автоматизации

Все делается аналогично созданию COM-объекта *(см. разд. 1.3)*. После выбора пункта **Automation Object** (Объект автоматизации) появится диалоговое окно мастера создания объекта автоматизации **Automation Object Wizard** (рис 1.3). Так же, как при создании COM-объекта, необходимо придумать название класса, указать способ создания объекта в раскрывающемся списке **Instancing** (Типы объекта) и потоковую модель — в списке **Treading Model**. Кроме того, флажок **Generate Event support code** позволяет указать, требуется ли генерировать код для поддержки обработки событий. Поскольку наш простейший объект не будет генерировать никаких событий, можно не изменять эту опцию, а также оставить и все прочие опции со значениями по умолчанию. После нажатия кнопки **OK** станет доступна библиотека типов нашего объекта. Действия по созданию интерфейсов и методов аналогичны действиям, описанным в *разд. 1.3*, с объектами COM.

После того как все действия выполнены, добавим в получившийся код реализацию метода `NextInt`. В листинге 1.11 приведен модуль реализации класса (каталог 11_OLEAutomation\Server).

Листинг 1.11. Модуль класса с реализованным интерфейсом (SimpleOLE.pas)

```
unit SimpleOLE;
interface
uses
  ComObj, ActiveX, ServerOLE_TLB, StdVcl;
```

```
type
  TSimple = class(TAutoObject, ISimple)
  protected
    procedure NextInt(aValue: Integer; out outValue: Integer); safecall;
    { Protected declarations }
  end;

implementation

uses ComServ;

procedure TSimple.NextInt(aValue: Integer; out outValue: Integer);
begin
 outValue:=aValue+1;
end;

initialization
  TAutoObjectFactory.Create(ComServer, TSimple, Class_Simple,
    ciMultiInstance, tmApartment);
end.
```

Теперь осталось сохранить проект, откомпилировать код и зарегистрировать новый OLE Automation Server. Регистрация производится либо так же, как и регистрация COM-объектов, либо выбором пункта **Run | Register ActiveX Server** (Запуск | Зарегистрировать ActiveX Server), либо вручную через системную утилиту Regsvr32.exe. Чтобы зарегистрировать сервер вручную, файл DLL следует скопировать в системную папку операционной системы. Для Windows 98 — это каталог C:\Windows\System\, для Windows 2000 и Windows XP — X:\%Windir%\System32\. Однако можно и не копировать библиотеку в системную папку, в этом случае при регистрации необходимо указать полный путь к файлу.

Обратимся к системной утилите Regsvr32.exe, чтобы зарегистрировать новый сервер. Из командной строки запустите утилиту с одним параметром — именем файла сервера компонента:

```
>Regsvr32.exe ServerOLE.dll
```

Если сервер становится не нужен, его можно снять с регистрации, дав команду запуска утилиты Regsvr32.exe с ключом `/u`:

```
>Regsvr32.exe /u ServerOLE.dll
```

Замечу, что помимо ключей, характерных для COM-объектов, серверы автоматизации создают еще один ключ с параметром `ProgID` вида:

```
HKEY_CLASSES_ROOT\{ProgID string}
```

который содержит `CLSID` сервера автоматизации и позволяет по перекрестной ссылке вернуться к первой ссылке в разделе `HKEY_CLASSES_ROOT\CLSID`. В данном случае строка `ProgID` будет иметь вид *Имя_DLL.Имя_объекта*, т. е. если проект сохранен под именем `SimpleOLE` (DLL будет иметь то же самое имя), а имя объекта есть имя, введенное при создании Automation Object — `ServerOLE`, то значение `ProgID` равно `SimpleOLE.ServerOLE`. Поэтому, зарегистрировав сервер автоматизации, вы можете найти данный ключ у себя в реестре. Однако не все информативные имена серверов составляются по такому принципу. Мы рассмотрим один из способов определения имени в случае объекта MS Agent и других объектов.

Код приложения-клиента с ранним связыванием ничем не будет отличаться от кода вызова методов COM-объектов. Этот код находится в подкаталоге 11_OLEAutomation\EarlyClient. Код приложения-клиента с поздним связыванием несколько другой — он приведен в листинге 1.12).

Листинг 1.12. Клиент с поздним связыванием (каталог 11_OLEAutomation\LateClient)

```
unit Unit1;
interface
uses
  ..., ComObj;

type
  TForm1 = class(TForm)
    ...
  end;

var
  Form1: TForm1;
  V: Variant;
  i: integer;

implementation
{$R *.DFM}
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 V:=CreateOleObject('ServerOLE.Simple');
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 i:=StrToInt(Edit1.Text);
 V.NextInt(i, i);
 Edit1.Text:=IntToStr(i);
end;

end.
```

1.5. Технология ActiveX в среде Delphi

В этом разделе мы кратко познакомимся с технологией ActiveX, создадим простейший ActiveX-компонент, внедрим его в операционную систему и попытаемся им воспользоваться.

Краткие сведения

По-разному можно ответить на вопрос "Что такое ActiveX?" Все зависит от того, с какой точки зрения посмотреть. Для программистов, работающих в таких средах, как Delphi или Visual Basic, ActiveX — это "черный ящик", содержащий свойства, методы и события. А с точки зрения модели COM элемент управления AcitveX — это сервер автоматизации, реализованный в виде DLL, исполняемый в одном процессе с вашим приложением и допускающий визуальное редактирование.

Многие привычные компоненты Delphi, расположенные на его палитрах, представляют собой не что иное, как стандартные элементы управления ActiveX операционной системы.

Процесс разработки компонентов ActiveX "с нуля" настолько сложен и трудоемок, что ни опытные программисты, ни тем более новички обычно не создают их напрямую. В среде Delphi существует несколько мастеров, с помощью которых возможно превратить в элемент AcitveX любой визуальный компонент, а также создать активный элемент из формы.

В последующих разделах мы рассмотрим приемы создания элементов управления на базе ActiveForm — так называется технология превращения форм в компоненты ActiveX.

Создаем свой элемент ActiveX

Определимся с постановкой задачи, сделав перед этим небольшое отступление философского характера. Нередко тем, кто создает текстовые редакторы, приходится заниматься написанием однородных пользовательских интерфейсов и однообразных программных кодов. Описание реализации одних и тех же функций — открытия, сохранения файлов, копирования, вставки и т. д. — может порядком наскучить. Проблема, конечно, не животрепещущая, однако будет интересно показать, как можно ее решить, пользуясь технологией ActiveX и в частности — ее вариацией ActiveForm.

Итак, определимся: создаваемая форма должна иметь одну или несколько панелей управления со стандартными кнопками ("создать", "открыть", "сохранить", "вырезать", "копировать", "вставить" и др.), а также многострочный текстовый редактор. Пример подобной формы приведен на рис. 1.4.

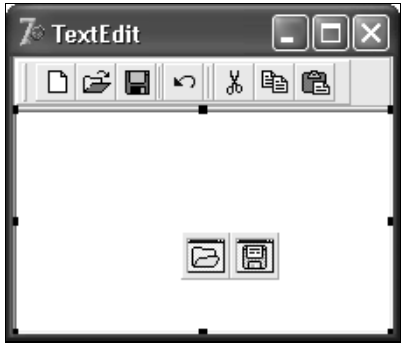


Рис 1.4. Пример формы для будущего ActiveX-компонента

В среде Delphi в пункте **New** (Новый) меню **File** (Файл) на уже знакомой нам вкладке **ActiveX** выбираем элемент **ActiveForm**. Появится диалоговое окно мастера, в котором следует ввести имя компонента (**New ActiveX Name**), название модуля реализации (**Implementation Unit**), название проекта (**Project Name**) и тип модели — модель потока (**Threading Model**). Естественно, что скомпилированная динамическая библиотека будет иметь то же название, что и название проекта, однако с расширением не dll, а ocx.

Назовем компонент TextEditX, и одновременно с этим модуль реализации автоматически переименуется в `TextEditImpl1.pas`, а проект — в `TextEditProj1.dpr`. После нажатия кнопки **OK** появится заготовка кода

проекта, окно формы и все необходимые модули, а также к проекту добавится библиотека типов `TextEditProj1_TLB.pas`.

Новую пустую форму необходимо доработать до требуемого вида, т. е. перенести на нее необходимые компоненты и описать реакцию на требуемые события.

На этом можно было бы закончить пояснения, т. к. остается только скомпилировать проект и заняться установлением элемента управления в систему, однако осталась невыясненной одна деталь. Конечный разработчик нашей формы должен будет через дополнительные свойства узнавать текст, содержащийся в многострочном текстовом редакторе (ради чего мы и создавали компонент ActiveX), а также должен иметь возможность получать информацию о каких-либо специфических свойствах компонента. Немаловажно также будет иметь средства извещения приложений и о том, что в компоненте произошли какие-то изменения — например, текста.

В связи с этим мы продолжим совершенствовать наш компонент ActiveX:

1. Откройте Редактор библиотеки типов TextEditProj1_TLB.pas.
2. Выделите интерфейс `ITextEditX` в иерархическом списке, затем добавьте в него новое свойство для чтения и для записи с именем `Text`. В списке оно отобразится в виде пары свойств — одно для чтения, другое для записи.
3. Обновите код. В модуле описания формы появятся две заготовки для описания реализации двух функций. Опишем функции:

```
function TTextEditX.Get_Text: OleVariant;
begin
  Result:=reEdit.Lines.Text;
end;

procedure TTextEditX.Set_Text(Value: OleVariant);
begin
  reEdit.Lines.Text:=VarToStr(Value);
end;
```

4. Добавьте новое событие `OnTextChanges` в иерархический список к интерфейсу `ITextEditXEvents`.
5. Теперь наша задача — вовремя инициировать наступление этого события, т. к. реализацию буду писать программисты, пользующиеся нашим компонентом. А потому код генерации события необходимо вставить в обработчик события изменения текста в многострочном редакторе. Дважды

щелкните в области окна редактора, после чего откроется заготовка процедуры реализации события. Туда следует поместить следующий код:

```
procedure TTextEditX.reEditChange(Sender: TObject);
begin
 if FEvents<>nil then
   FEvents.OnTextChanges;
end;
```

На этом разработку элемента управления можно считать законченной. Приступим к его регистрации и использованию.

Внедрение и использование компонента ActiveX

После компиляции библиотеки в папке с проектом появится готовый элемент ActiveX; в нашем случае соответствующий файл будет называться TextEditProj1.ocx. Элемент управления можно зарегистрировать обычным способом через пункт меню, но можно сделать это вручную через системную утилиту Regsvr32.exe. Таким же способом можно регистрировать обычные COM-объекты и OLE-серверы программным путем, например, в инсталляторе. Правда, и сам Delphi точно так же регистрирует наши собственные компоненты.

Осталось подключить элемент управления к среде Delphi. В меню **Component** (Компоненты) выберите пункт **Import ActiveX Control** (Импорт элемента управления ActiveX). Откроется диалоговое окно **Import ActiveX** (Импортирование ActiveX) со списком доступных компонентов ActiveX (рис. 1.5).

Найдите в списке доступных компонентов наш элемент управления, нажмите кнопку **Install** (Установить), в появившемся окне установки компонента перейдите на вкладку **Into new package** (В новый пакет), в поле **File name** (Имя файл) укажите папку для сохранения пакета (рис. 1.6) и по желанию добавьте к нему описание в поле **Description** (Описание). После щелчка на кнопке **OK** откроется окно пакета, куда будет установлен новый ActiveX-компонент. Соответствующая ему пиктограмма появится на палитре компонентов **ActiveX**, и об его установке будет сообщено в диалоговом окне. После этого закройте окно пакета (рис. 1.7), сохранив его по запросу.

Приложение текстового редактора будет состоять всего лишь из одного компонента ActiveX (см. пример приложения из каталога 12_3_AppActiveX). На рис. 1.7 представлено изображение главной формы приложения во время разработки.

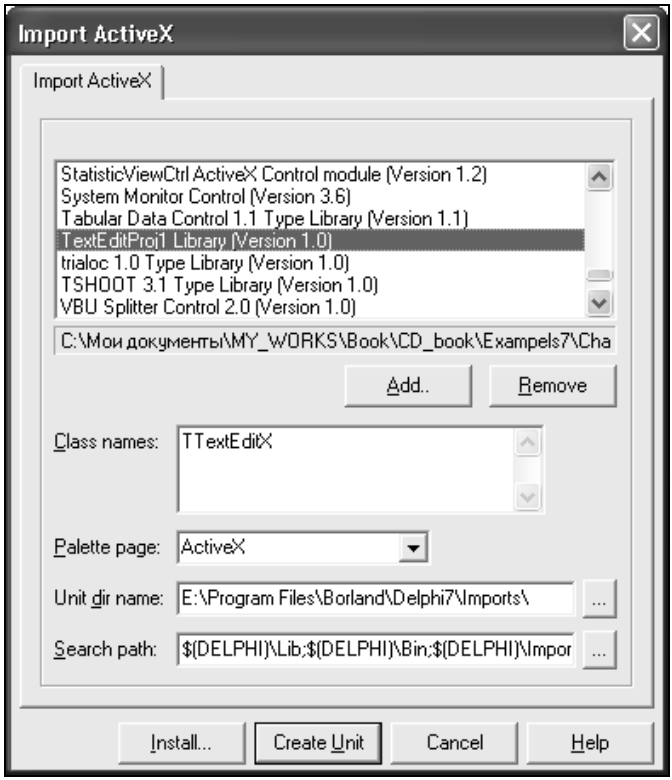


Рис. 1.5. Диалоговое окно импорта элемента управления ActiveX

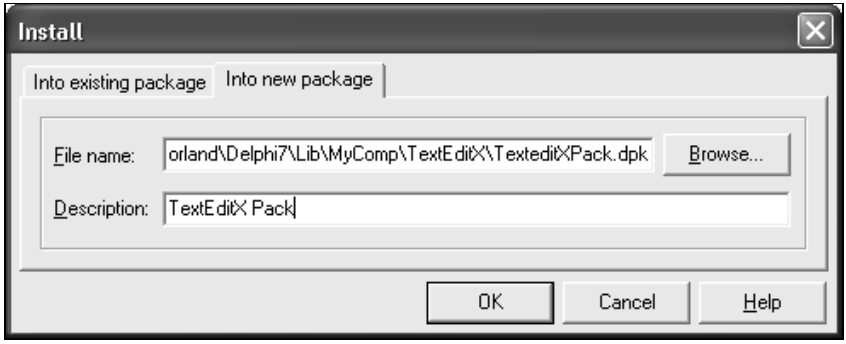


Рис. 1.6. Окно установки компонентов в новый пакет

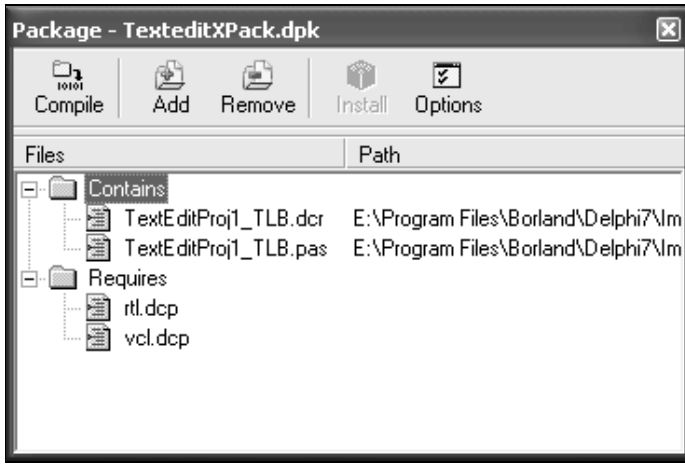


Рис. 1.7. Окно нового пакета
с установленным элементом управления `TextEditXMS`

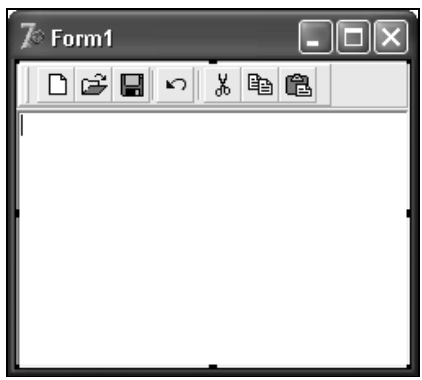


Рис. 1.8. Главная форма приложения
во время разработки

Естественно, данный элемент ActiveX с таким же успехом можно применять и в коде HTML с помощью тега `<OBJECT>`, и таким же образом можно описывать обработчики событий на VBScript или на JavaScript. В листинге 1.13 приведен HTML-код с внедренным элементом управления, в котором реализована обработка события `OnTextChanges` на языке VBScript.

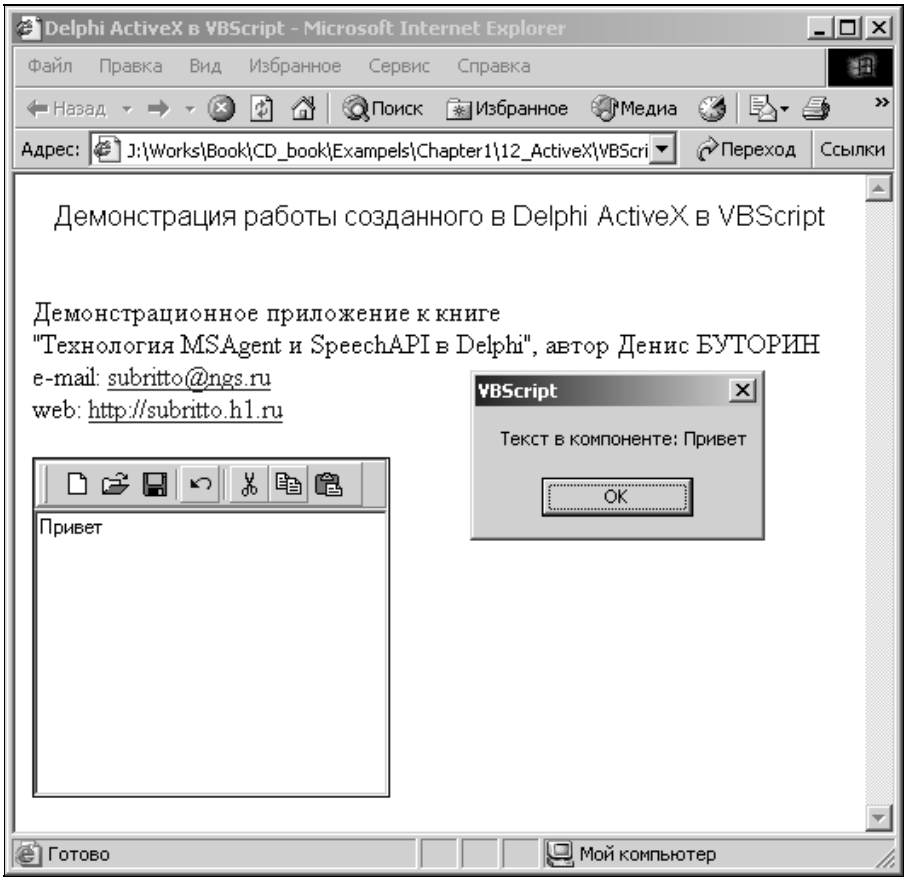


Рис. 1.9. Пример работы созданного в Delphi компонента ActiveX на Web-странице

Листинг 1.13. Пример HTML-кода с элементом ActiveX из Delphi (скрипт на VBScript)

```
<html>
 <head>
 <title>Delphi ActiveX в VBScript</title>
 <meta http-equiv="Содержимое-Тип" content="text/html; charset=windows-
1251">
 </head>
 <body bgcolor="#FFFFFF" text="#000000">
```

```
 <p align="center">
   <font face="Arial, Helvetica, sans-serif"
      size="3">Демонстрация работы созданного в Delphi ActiveX
      в VBScript
   </font>
 </p>
 <p>
   <object width="210" height="200"
      classid="CLSID:790700C5-4CC5-11D8-8903-0020ED19BE94" id="TE">
   </object>
 </p>
 <script language="VBScript">
   sub TE_OnTextChanges
         i = TE.Text
         MsgBox "Текст в компоненте: "&i
   End sub
 </script>
 </body>
</html>
```

В каталоге 12_ActiveX\VBScript вы найдете файл script.htm, работу которого можно посмотреть в любом браузере, поддерживающем ActiveX, например, в Internet Explorer (рис. 1.9).

Если у вас ничего не получится, т. е. на странице ничего не будет видно, то попробуйте разрешить загрузку неподписанных элементов ActiveX в диалоговом окне **Параметры безопасности** (Security Settings), открываемом щелчком на кнопке **Другой** (Custom Level) вкладки **Безопасность** (Security) диалогового окна **Свойства обозревателя** (Internet Properties).

1.6. Резюме

Итак, мы рассмотрели в общих чертах технологию Component Object Model. Мы попытались понять заложенные в ней принципы программирования и узнали, что она является основой других замечательных технологий, таких как ActiveX. Подытожим полученные знания.

В технологии COM важно выделить несколько базовых элементов: объект, интерфейс, сервер, библиотека COM, библиотека типов и др.

Приложение COM включает в себя один или несколько объектов COM, которые обладают некоторым набором интерфейсов. Интерфейсы имеют не

только имя, но и уникальный идентификатор. Каждый интерфейс обладает методами и реализуется в определенном классе Delphi. Получив один указатель на какой-либо интерфейс объекта, можно получить указатели на остальные интерфейсы через него.

При отсутствии указателя на какой бы то ни было интерфейс объекта необходимо обратиться к библиотеке COM через функцию `CoCreateInstance`, передав при этом идентификатор класса, тип сервера и идентификатор интерфейса. Затем с помощью имеющегося указателя на интерфейс объекта можно получить указатели на остальные интерфейсы через метод `QueryInterface`.

В целях обеспечения возможности работы с COM-объектами из интерпретируемых языков разработана технология OLE Automation. На OLE-объекты налагаются некоторые ограничения, вследствие чего можно организовывать позднее связывание с этими объектами — вызывать методы объекта без знания таблицы адресов методов (иначе говоря, без описаний интерфейсов).

Ключевую роль в позднем связывании играют методы `Invoke` и `GetIDsOfNames` интерфейса `IDispatch`. При трансляции программного кода вызов метода автоматизации через тип `Variant` (`OleVariant`) заменяется получением идентификатора метода (`DispID`) по его имени с помощью метода `GetIDsOfNames`. Затем по идентификатору вызывается необходимый метод объекта через метод `Invoke`.

Технология ActiveX является продолжением технологий COM и OLE. Элемент управления ActiveX представляется для разработчика "черным ящиком", содержащим свойства, события, методы. Он допускает визуальное редактирование в среде разработки, т. е. чем-то напоминает библиотеку визуальных компонентов VCL Component из Delphi.

Глава 2

Технология MS Agent в среде Delphi

Эта глава посвящена знакомству с Microsoft Agent и программированию анимированных персонажей в среде Delphi. Первым делом мы узнаем, что представляет собой технология анимированных персонажей, затем рассмотрим компоненты пользовательского интерфейса и определим основные правила (или рекомендации), придерживаясь которых при создании агента вашего приложения, вы не сделаете его лишней "фишкой". Здесь впервые персонаж появится на экране.

Затем мы перейдем к программированию. Подробно поговорим об объектной модели MS Agent, ее основных интерфейсах, свойствах, событиях, методах. Научимся пользоваться персонажем, а также управлять поведением нескольких персонажей одновременно и многому другому.

2.1. Что такое MS Agent?

Для того чтобы научиться создавать приложения с использованием технологии MS Agent, необходимо разобраться, что она из себя представляет. После этого нужно установить необходимые компоненты, затем научиться работать с персонажем как пользователь, а уже потом перейти к программированию. Так и поступим.

Microsoft Agent

Microsoft Agent — это сервер автоматизации (OLE Automation Server), который позволяет расширить форму взаимодействия приложения или web-узла с пользователем с помощью интерактивных анимированных персонажей в рамках интерфейса Microsoft Windows. Разработчики могут использовать данный компонент для расширения пользовательского интерфейса, в который входят такие привычные элементы, как кнопки, меню, панели управления и т. п.

Иными словами, Microsoft Agent — служба операционной системы, которая позволяет встраивать анимированные персонажи в приложения и web-страницы. Этот сервис позволяет программно обращаться к нему из других приложений, причем служба MS Agent допускает вызов одновременно несколькими клиентами.

Все знают забавного Помощника в пакете Microsoft Office — Скрепку, Мурку и т. д. Так вот, эти персонажи — это результат применения технологии Microsoft Agent в конечном приложении (виды различных персонажей MS Agent показаны на рис. 2.1).

Рис. 2.1. Виды различных персонажей MS Agent.

В Microsoft Agent реализовано усиление естественных для человека способов взаимодействия с приложением. Кроме стандартных принципов ввода информации с помощью клавиатуры и мыши, MS Agent включает поддержку распознавания речи. В дополнение к этому персонаж может отвечать пользователю, используя синтез речи или проигрывание записанного аудиофрагмента, а также с помощью текста, появляющегося в "воздушном шарике" рядом с "говорящим" персонажем.

MS Agent не заменяет стандартного (оконного) графического пользовательского интерфейса (Graphical User Interface — GUI). И может показаться странным тот факт, что, применяя анимированный персонаж, разработчик небольших приложений действительно способен отказаться от обычного типа интерфейса и реализовывать все органы взаимодействия с пользователем через MS Agent. Однако эта идея абсолютно реалистична! Например, вы пишете почтовую программу, заменяя привычный интерфейс персонажем-агентом, а все команды меню и кнопочных форм заменяя на речевые, подаваемые агенту голосом. Персонаж распознает[1] их и выполняет необходимые действия! Вы говорите: "Receive mail!", и Помощник запускает подключение к Интернету, соединяется с сервером и принимает почту. Кроме того, агент может проинформировать вас обо всех событиях, происходящих в приложении, голосом,

[1] Конечно, не сам персонаж распознает речь, а другие компоненты, о которых мы поговорим позднее.

вместо того чтобы открывать диалоговые окна. Допустим, ваш почтовый ящик пуст. Тогда агент сообщит: "В вашем ящике нет новых писем!"

Приложение, пользующееся компонентом MS Agent, получает доступ к персонажу (ассистенту), списку его анимаций, командам ввода/вывода (в том числе и речевым). Как и любой COM-объект, Microsoft Agent автоматически запускается, как только хотя бы одно приложение начинает его использовать, и выгружается, когда отпадает необходимость в последней ссылке на него.

Не следует путать службу управления MS Agent с самим персонажем. Компонент сервиса операционной системы представляет собой набор динамических библиотек и исполняемых файлов, которые реализуют все необходимые функции. Ассистенты, появляющиеся на экране и обладающие набором анимаций, находятся в отдельных файлах. Они загружаются с помощью специальных функций, после чего набор анимаций, реализованный в персонаже, становится доступным для проигрывания. Устанавливаются ассистенты отдельно, независимо от сервиса Microsoft Agent.

MS Agent поддерживает два формата файлов персонажей и данных анимации: простой структурированный файл (тип ACS) и раздельный файл (типы файлов ACF для данных и ACA для анимаций). Создаются персонажи в специальной утилите Microsoft Agent Character Editor (о ней мы поговорим в *гл. 4*). Чаще используются файлы формата ACS, т. к. в них хранятся все анимации персонажа. Раздельный формат лучше всего применять, когда есть желание загружать анимации персонажа отдельно, например, с HTTP-сервера. В таком случае при запросе анимации данные будут загружены с web-сервера, однако при повторном вызове анимации браузер проверит кэш, и если анимация уже загружалась, то данные будут воспроизведены из кэша.

Синтез и распознавание речи также не реализованы в самом компоненте MS Agent, для подключения этих функций сервер использует другие компоненты операционной системы, такие как пакет функций Speech API, речевые синтезаторы (движки, engine) и модули распознавания речи. Где их найти, как установить и каким образом ими пользоваться, мы узнаем позднее, как только эти функции нам понадобятся, пока же обратимся к MS Agent.

Установка компонентов и импорт ActiveX

Прежде чем обращаться к интерфейсам сервера Microsoft Agent, его нужно установить в операционную систему. В Windows 2000/XP (и более поздних версий) данный компонент установлен по умолчанию, однако пользователям Windows 98 необходимо самостоятельно инсталлировать его. Вместе с

MS Office 2000 устанавливается Microsoft Agent v1.5, однако мы будем работать с версией 2.0. Поэтому, даже если у вас в Windows 98 установлен пакет Microsoft Office 2000, вам все равно придется установить компонент Microsoft Agent дополнительно. Установка новой версии никак не скажется на работоспособности стандартных персонажей MS Office, так что не надо беспокоиться.

Дистрибутив MS Agent можно свободно загрузить с сайта Microsoft по адресу **http://activex.microsoft.com/activex/controls/agent2/MS Agent.exe**. Он представляет собой самоустанавливающийся CAB-архив объемом в 400 Кбайт. Все ссылки на необходимые модули вы найдете в *приложении 1.*

Технология MS Agent является продолжением Microsoft Windows и, как следствие, поддерживается только в операционных системах от Windows 95 и выше. Установка компонентов MS Agent требует наличия в системе некоторых необходимых библиотек, таких как Microsoft Visual C++ Runtime Library (Msvcrt.dll), Microsoft Registration Tool (Regsvr32.exe), и Microsoft COM. Наилучший способ обеспечить существование этих библиотек, а также их версий, которые могут потребоваться, состоит в том, чтобы установить Internet Explorer версии не ниже 4.0.

После запуска инсталлятора компонента Microsoft Agent появится окно с лицензией Microsoft. Если мы хотим установить MS Agent, нам ничего не остается делать, кроме как согласиться с лицензионными требованиями. После этого начнется копирование файлов, которое не займет много времени.

MS Agent устанавливается в каталог системы \%SystemRoot%\MS Agent\, где \%SystemRoot%\ — папка операционной системы. По умолчанию для Windows 98 это C:\Windows, для Windows 2000 — x:\winnt, для Windows XP — так же x:\windows.

Думаю, большей части читателей роднее русский язык, и управлять элементами пользовательского интерфейса MS Agent на национальном языке им будет приятнее. Поэтому не лишним станет установка модуля Microsoft Agent International DLL Russian. Все реализуемые в MS Agent диалоги, подписи, а также сообщения об ошибках будут после этого "идти" на русском. Национальные модули существуют почти для всех распространенных языков — в списке их около 30 и среди них присутствуют: арабский, греческий, датский, итальянский, китайский, корейский, немецкий, русский, словакский, словенский, тайский, турецкий, финский, французский, хорватский, чешский, японский и т. д. Эти модули можно также скачать с официального сайта Microsoft. Национальные модули устанавливаются в папку \%SystemRoot%\MS Agent\Intl\.

Поскольку компонент Microsoft Agent является компонентом системы, он не может быть удален из нее. То же действительно и для тех операционных систем Windows, в которых MS Agent не присутствует как компонент по

умолчанию, — устанавив данный сервис с дистрибутива, вы не сможете впоследствии удалить его.

Итак, MS Agent установлен. Однако, как отмечалось ранее, сам компонент не может работать без установленных персонажей. Следовательно, хотя бы один из них необходимо установить, поэтому можете смело скачивать их из Интернета *(см. приложение 1)*. Некоторые персонажи распространяются в инсталляционных файлах, представляющих собой самоустанавливающийся CAB-архив, и в данном случае они ничего не записывают в реестр, а просто распаковывают файл персонажа в указанную папку. Каталог с персонажами располагается в подкаталоге MS Agent: \%SystemRoot%\MS Agent\Chars\. Поэтому, если вы раздобыли в Интернете файл персонажа (файл в формате ACS), то скопируйте его в данную папку. Вообще говоря, персонажи могут, конечно, располагаться в любом каталоге на жестком диске, однако удобнее всего хранить их в папке \MS Agent\Chars. При загрузке персонажей необходимо указывать имя файла ассистента, а если он лежит не в папке \Chars — то полный путь к нему.

С установкой Microsoft Agent мы покончили. Теперь все приложения, использующие функции MS Agent (за исключением синтеза и распознавания речи), будут исправно функционировать. Для подключения речевых возможностей следует установить еще ряд дополнительных компонентов. Рассматривать мы их будем в следующих главах, а пока скажу только, что для проверки и демонстрации речевых способностей MS Agent необходимы: пакет функций Microsoft Speech API v4.0, хотя бы один речевой синтезатор (движок) и модуль распознавания речи. Синтезаторы речи существуют для большого круга языков, а вот с распознаванием сложнее, и нам придется ориентироваться на английскую речь (ну, или на японскую, кому какая больше нравится ;-)). Движков распознавания русской речи в "сыром" виде (т. е. в виде дистрибутивов) я нигде не встречал. Обычно отечественные системы распознавания речи скрывают свои двигатели от свободного использования их другими программами.

Клиентские приложения могут обращаться к Microsoft Agent Services напрямую, используя интерфейсы сервера автоматизации (Agent Server), однако Microsoft Agent включает в себя элемент управления ActiveX — Microsoft Agent Control, обеспечивающий наиболее простой способ программирования во всех средах, поддерживающих интерфейс управления ActiveX, а также на языках сценариев типа Visual Basic, JavaScript. Поэтому нелишним будет заранее установить данный компонент в среду программирования, ведь рано или поздно мы начнем использовать MS Agent в собственных приложения.

Запустите Delphi, закройте пустой шаблон приложения, в меню **Component** (Компонент) выберите пункт **Import ActiveX Control** (Импортировать элемент управления ActiveX), в появившемся диалоговом окне **Import ActiveX**

(рис. 2.2) в списке доступных для импорта элементов управления найдите пункт **Microsoft Agent Control 2.0**. Если же его в списке нет (хотя вы устанавливали компонент MS Agent), то щелкните на кнопке **Add** (Добавить), найдите в открывшемся диалоговом окне открытия файлов библиотеку Agentctl.dll в папке \%SystemRoot%\MS Agent\ и добавьте ее к списку.

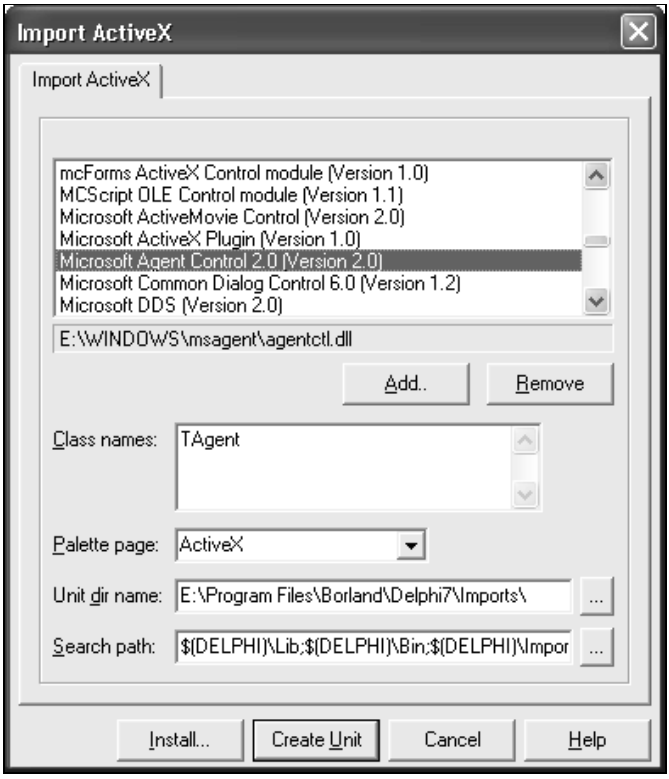


Рис. 2.2. Диалоговое окно импорта элемента управления ActiveX

После выбора ActiveX в списке доступных элементов управления щелкните на кнопке **Install** (Установить). В появившемся одноименном диалоговом окне следует выбрать пакет для установки элемента управления. По умолчанию это пакет Borland User Components, туда мы и установим новый элемент управления ActiveX (рис. 2.3).

Щелкните на кнопке **OK**, и после непродолжительной перекомпиляции в пакет будет добавлен новый AcitevX-компонент Microsoft Agent Control. Пиктограмма элемента управления — портрет своеобразного господина в шляпе и очках, настоящего "агента", — появится на палитре компонентов ActiveX. Скриншот части палитры компонентов приведен на рис. 2.4.

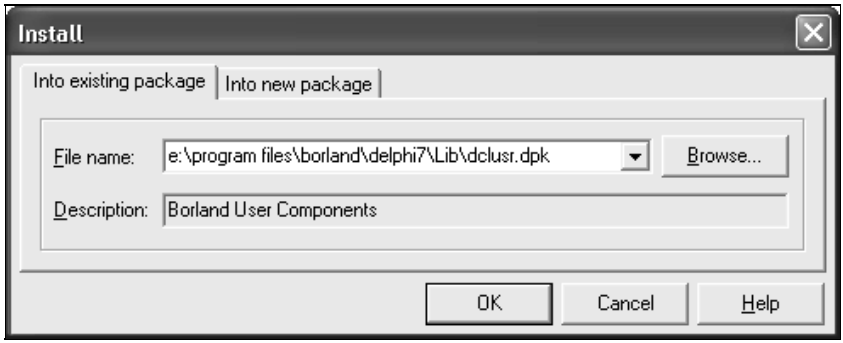


Рис. 2.3. Диалоговое окно выбора пакета для установки нового элемента управления ActiveX

Рис. 2.4. Часть палитры компонентов с добавленным элементом Microsoft Agent Control

Важный совет

Приведенные в книге примеры с использованием MS Agent рассчитаны на все версии Delphi (от 5 до 7), поскольку MS Agent работает во всех версиях Windows и Delphi. Однако, как я заметил в процессе написания статей для авторской web-страницы (еще до написания данной книги), в общей массе пользователей Delphi 7 был не в ходу. Но вскоре ситуация изменилась и посетители сайта стали отмечать одну особенность, а именно то, что "под Delphi 7" некоторые примеры некорректно работали (скажем, примеры с поиском анимаций персонажа и т. п.) Вскорее я также стал переходить на седьмую версию любимой среды и на своем компьютере отметил точно такаю же ситуацию. Я долго ломал голову над этой проблемой — переустанавливал среду, MS Agent — но ничего не помогало. Как-то мне пришла в голову мысль проверить работоспособность примеров с файлами библиотек типов от Delphi 5. И действительно, как только я заменил файлы AgentObjects_TLB.pas и AgentServerObjects_TLB.pas, созданные Delphi 7, на файлы от Delphi 5, все проекты заработали как надо. Эта новость появилась на моей web-странице, и через некоторое время посетители отметили, что данное решение сняло проблему. В связи с этим на прилагаемом компакт-диске сосуществуют модули от разных версий Delphi, и если у вас установлен Delphi 7 и некоторые проекты, взятые с компакт-диска, будут выдавать ошибки при выполнении — обязательно попробуйте заменить заголовочные файлы на модули, созданные Delphi 5 или 6. Естественно, не забудьте сохранить свои. Ну, а если что-то все-таки не будет работать, то обращайтесь ко мне по e-mail: **subritto@ngs.ru** или задавайте вопросы в гостевой книге и на форуме моей страницы: **http://subritto.h1.ru**.

Лицензия на использование и распространение

Это, пожалуй, самый неприятный и нудный раздел, но с ним также следует ознакомиться. Эта тема станет особенно актуальна и к ней придется вернуться тогда, когда читатель перейдет к созданию своих программ, их распространению, а также к переводу их на коммерческую основу.

Здесь я приведу несколько выдержек из общей лицензии на распространение программных продуктов фирмы Microsoft®, в том числе касающихся и Microsoft Agent®. Полностью ознакомиться с лицензией можно по адресам:

http://www.microsoft.com/rus/permission/copyrgt/cop-soft.asp

http://www.microsoft.com/rus/permission/copyrgt/cop-img.asp

Использование программного обеспечения

Программы для бесплатной загрузки

Корпорация "Майкрософт" предлагает ряд своих продуктов и программных компонентов (далее — Программное обеспечение), включая элементы программного обеспечения и системные файлы, для бесплатной загрузки через Интернет. Хотя вы можете свободно загружать и использовать Программное обеспечение, вы не имеете права воспроизводить и повторно распространять его. Чтобы определить свои права на повторное распространение, обратитесь к лицензионному соглашению, поставляемому вместе с Программным обеспечением. Если ваши права на повторное распространение не оговорены в лицензионном соглашении прямо, распространение Программного обеспечения запрещено. Вы можете сообщать конечным пользователям адреса страниц загрузки на нашем web-узле для получения Программного обеспечения непосредственно от корпорации "Майкрософт". Если вы хотите опубликовать на своем web-узле наш URL-адрес, ознакомьтесь с указаниями по использованию ссылок: **http://www.microsoft.com/rus/permission/copyrgt/cop-img.asp#LinkTo**.

Файлы CAB

Корпорация "Майкрософт" не предоставляет права на распространение отдельных файлов CAB. Конечные пользователи должны получать эти файлы с оригинальными компакт-дисками продуктов корпорации "Майкрософт".

Передача лицензии

Бесплатное предоставление или передача третьей стороне программного продукта, лицензий на программное обеспечение или аппаратных средств, на которых установлены программные продукты корпорации "Майкрософт",

может производиться без письменного разрешения, если соблюдаются условия соответствующего лицензионного соглашения. Любая законная передача, будь то торговая сделка, пожертвование или подарок, должна включать предоставление документации, руководств, оригинальных дисков и лицензий для продукта. В соответствующем лицензионном соглашении могут быть оговорены дополнительные правила передачи. Частное лицо или организация, осуществляющие передачу программного обеспечения и лицензии (лицензий), должны отдавать себе отчет в том, что они отказываются от всех прав на использование соответствующего программного обеспечения, включая все права на получение обновленных версий продукта.

Авторские права: использование изображений

Персонажи Microsoft Agent, помощники Office и другие персонажи, поставляемые с продуктами корпорации "Майкрософт"

Персонажи Microsoft Agent. Согласно условиям лицензионного соглашения для Microsoft Agent, разработчики имеют право реализовывать изображения персонажей, поставляемые корпорацией "Майкрософт", ТОЛЬКО через интерфейс программирования Agent, а также, если требуется помочь пользователям в выборе персонажа, включать эти изображения (без изменения) в приложения и страницы с поддержкой сценариев, где применяется технология Agent. При этом должно присутствовать уведомление "С использованием технологии Microsoft Agent", а также уведомление об авторском праве, как изложено в лицензионном соглашении для Microsoft Agent. Если разработчик использует Microsoft Agent, он не обязательно обладает неограниченными правами на использование изображений Microsoft Agent. Как указано в лицензионном соглашении для Microsoft Agent, запрещается каким-либо образом изменять персонажи, поставляемые корпорацией "Майкрософт". В частности, запрещается изменять имя персонажа, определенное корпорацией "Майкрософт", или включать имя персонажа в название собственного продукта, поскольку это создаст неправильное представление о праве собственности на персонаж.

ТЕМ НЕ МЕНЕЕ, разрешается использование персонажей Microsoft Agent в экранных снимках при условии, что изображения персонажей не изменяются и являются элементом снимка окна программы, иллюстрирующей работу технологии Microsoft Agent. Обратите внимание на то, что правила использования экранных снимков действуют наравне с ограничениями использования, изложенными в соглашении для Microsoft Agent. При использовании экранных снимков также требуется включать уведомления об авторских правах для каждого персонажа Microsoft Agent. Категорически запрещается использовать изображения персонажей, поставляемые корпорацией "Майкрософт", в качестве элементов оформления или эмблем других продуктов.

**Примечание. Указанные условия и ограничения распространяются только на изображения персонажей, поставляемые самой корпорацией "Майкрософт" для использования с программой Microsoft Agent. Условия использования любых других персонажей (авторские права на которые не принадлежат корпорации "Майкрософт") определяются соответствующим владельцем. Разработчики, создающие собственные уникальные персонажи с помощью средства Microsoft Agent Character Editor, могут использовать, распространять или лицензировать их по собственному усмотрению.

Помощники Office. Помощники Office могут использоваться только в качестве встроенного элемента или функциональной возможности программного продукта корпорации "Майкрософт". Любое другое использование помощников Office запрещено.

ИСКЛЮЧЕНИЯ. Использование изображений помощника Office, являющихся частью экранных снимков продуктов корпорации "Майкрософт", допускается в обучающих материалах и документации по продуктам корпорации "Майкрософт". Такие экранные снимки должны использоваться для иллюстрирования функции, выполняемой помощником Office в программном обеспечении корпорации "Майкрософт", но не в качестве средства оформления или графического элемента для каких-либо других целей. Кроме того, любое изменение помощников Office запрещается; помощники Office должны отображаться так же, как в программных продуктах корпорации "Майкрософт". Ознакомьтесь с правилами использования экранных снимков.

Другие персонажи,
поставляемые с продуктами корпорации "Майкрософт"

На персонажи, поставляемые с продуктами корпорации "Майкрософт", распространяется действие исключительной лицензии. Любое использование этих персонажей третьими лицами запрещено.

Компоненты пользовательского интерфейса

Чтобы познакомиться с персонажем и его компонентами пользовательского интерфейса (органами управления), запустим какую-нибудь программу, использующую MS Agent. К примеру, выберем демонстрационный проект DemoAgent, расположенный на прилагаемом компакт-диске (рис. 2.5).

Персонаж MS Agent отображается в собственном окне, т. е. сам ассистент и есть окно. Проверить это можно с помощью любого менеджера процессов и окон. При запуске приложения, использующего MS Agent, кроме окна приложения, в системе создается несколько дополнительных окон, одно из них с названием класса *AgentAnim* и заголовком окна *Microsoft Agent* (это и есть сам персонаж).

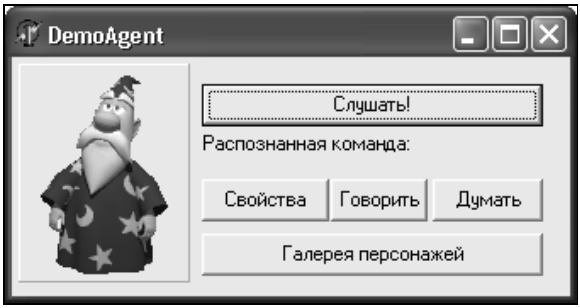


Рис. 2.5. Рабочее окно демонстрационного приложения DemoAgent

Персонаж можно перемещать по экрану, удерживая нажатой правую кнопку мыши. При щелчке правой кнопкой мыши на персонаже появляется контекстное меню со списком доступных команд, как показано на рис. 2.6.

Пункт **Открыть окно команд голосом** (Open Voice Commands Window)[2] появляется в том случае, когда приложение загрузило некоторые голосовые команды для персонажа. Следующий пункт — **Скрыть** (Hide) — отображается всегда, из одного этого пункта состоит простейшее контекстное меню персонажа, даже когда никаких команд нет.

После разделительной черты следуют пункты команд текущего активного клиента. Показанное на рис. 2.6 меню состоит всего из двух частей, однако далее может следовать еще одна разделительная черта и за ней — заголовки групп команд других приложений, которые используют тот же самый персонаж.

Вместе с загрузкой приложения и появлением персонажа в системном лотке SysTray (рядом с часами) появляется пиктограмма персонажа (рис. 2.7).

В подсказке, появляющейся при наведении мыши на пиктограмму, отображается имя персонажа на текущем языке операционной системы. Щелчок на пиктограмме вызывает появление персонажа на экране с проигрыванием анимации. Щелчок правой кнопкой мыши приводит к появлению меню, идентичного контекстному меню персонажа.

Следует отметить еще один элемент интерфейса — окно голосовых команд **Команды голосом**. Оно появляется при выборе пункта **Открыть окно команд голосом** контекстного меню персонажа или пиктограммы в системном лотке. На рис. 2.8 изображен пример окна команд голосом для персонажа Merlin (Мерлин).

[2] Английский вариант названия в случае, если распознается английская речь.

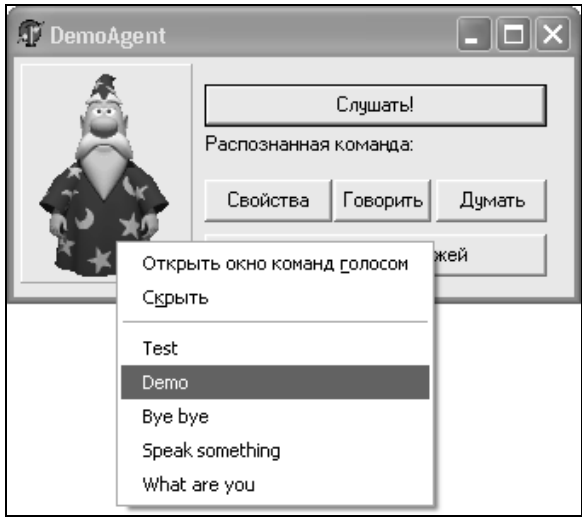


Рис. 2.6. Контекстное меню персонажа

Рис. 2.7. Пиктограмма MS Agent в SysTray (синий колпак со звездами)

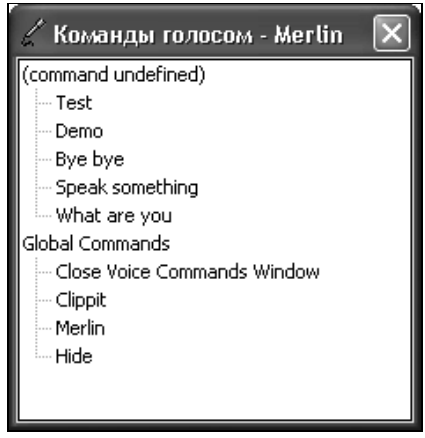


Рис. 2.8. Диалоговое окно команд голосом

Диалоговое окно команд голосом служит лишь подсказкой для пользователя при подаче команд персонажу, поскольку отсюда нельзя запустить их вы-

полнение. Все-таки это голосовые команды, и запускать их нужно именно голосом.

Вывод речи персонажа сопровождает выноска — "воздушный шарик" — с читаемым текстом, в стиле комиксов. Воздушный шар появляется, когда персонаж начинает говорить, и исчезает, когда его речь заканчивается. Программно, а также в диалоговом окне **Дополнительные параметры персонажей** (Advanced Character Options) (о котором сейчас пойдет речь), можно отключить отображение воздушного шарика и настроить другие атрибуты.

Когда персонаж "думает", т. е. выполняет действия, запускаемые специальным методом, текст "мыслей" появляется в воздушном шарике несколько иной формы (рис. 2.9).

Рис. 2.9. Слева показан шарик "говорящего" персонажа, справа —"думающего"

Если персонаж готов к принятию голосовых команд, он информирует об этом пользователя небольшим окошком с предупреждением (Tip Window). На рис. 2.10 изображен персонаж, готовый к принятию голосовых команд.

Рис. 2.10. Персонаж вместе с окошком-предупреждением Tip Window

Microsoft Agent поддерживает некоторые глобальные настройки, касающиеся всех персонажей. Они отражены в диалоговом окне **Дополнительные параметры персонажей** (рис 2.11), которое имеет три вкладки: **Вывод**, **Ввод речи** и **Авторские права**.

На вкладке **Вывод** можно задать скорость речи персонажа, включить воспроизведение звуковых эффектов и речи, а также определить необходимость дублирования читаемого текста в воздушном шарике (выноске).

Вкладка **Ввод речи** позволяет настроить параметры ожидания ввода речи от пользователя. В частности, настроить "горячую" клавишу, при нажатии на которую персонаж будет слушать голосовую команду, подачу звуковых сигналов по готовности MS Agent к принятию команд и т. п.

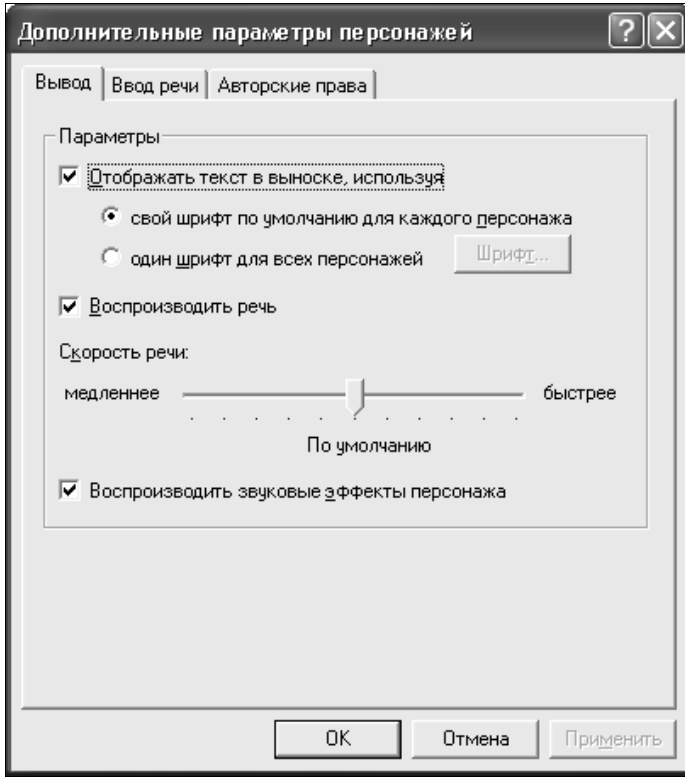


Рис. 2.11. Диалоговое окно **Дополнительные параметры персонажей**

Вкладка **Авторские права** для нас наименее интересна, поскольку и так понятно, что все права на данный продукт давно защищаются законодательством и принадлежат никому иному, как корпорации Microsoft.

Для установки персонажа по умолчанию существует еще одно диалоговое окно, похожее на окно выбора помощника в продуктах MS Office. В нем отображаются персонаж, его имя и краткая информация о нем. Пример этого диалогового окна дан на рис. 2.12.

Рис. 2.12. Диалоговое окно выбора персонажа

На этом обзор пользовательских органов управления MS Agent можно завершить. Как мы узнали, Microsoft Agent имеет достаточно разнообразные инструменты для управления персонажем и контроля его параметров. Да и сам персонаж настолько приятен и дружелюбен, что невозможно отказаться от его использования.

2.2. Советы по применению MS Agent в приложениях

В этом разделе мы рассмотрим основные принципы и методологию проектирования интерфейсов с использованием персонажей MS Agent. Каждый

разработчик должен понимать, что дает ему применение той или иной технологии и как добиться наибольшего эффекта или отдачи от использования новых компонентов программирования.

Поговорим об основных принципах взаимодействия персонажа с пользователем, о социальных аспектах и о деталях способов использования синтеза и распознавания речи.

Основные рекомендации

Проектируя интерфейс приложения с использованием анимированного персонажа, задумайтесь о том, какую функцию он будет нести и какую роль выполнять: помощника, советчика, руководителя, конферансье и т. п. Определите для себя также, на какую категорию пользователей будет рассчитано приложение.

Так как люди по своей природе, как правило, консервативны, может случиться, что не каждому пользователю понравится то обстоятельство, что по экрану будет разгуливать некое существо. Кроме того, если на персонаж "завязано" большое количество функций, к которым нет удобного доступа через привычный интерфейс, то при скрытии персонажа или при отсутствии необходимого для его функционирования компонента пользователь будет ощущать дискомфорт от использования данного продукта. Постарайтесь спроектировать приложение так, чтобы ассистент MS Agent являлся дополнением (расширением) стандартного интерфейса, и тогда пользователи оценят ваш вкус и изящество ваших решений. Ранее я отмечал, что возможно полностью заменить оконный интерфейс приложения виртуальным персонажем, однако это замечание было скорее данью восхищения технологией и возможностями анимированного персонажа, нежели советом по их применению.

Рекомендуется обратить внимание на обеспечение обратной связи с персонажем. Снабдите персонаж функциями отображения состояния приложения через выражение его лица или с помощью жестикуляции. Однако не забывайте и об уместности всех этих функций. Помните, что нельзя обманывать ожидания пользователя. Поэтому нелишним также будет использование соответствующей анимации при появлении и перед исчезновением персонажа, вместо мгновенного исполнения этих действий.

Не следует перенасыщать поведение персонажа активными действиями. Количество анимаций персонажа должно быть минимальным, чтобы не отвлекать пользователя от реальной работы. Вспомните — сильно ли вам надоедала Скрепка или Мурка (или кто там еще у вас живет?) в приложениях MS Office, пока вы набирали текст или обдумывали свои дальнейшие действия? Естественно, персонаж не должен "замораживаться", и тем не менее простая анимация типа дыхания, покачивания головой и т. п. создает иллю-

зию социального контакта с персонажем, демонстрирует его доступность и расположение к пользователю. Однако активная анимация хороша, когда необходимо привлечь внимание пользователя. Тут, конечно, персонажу нет равных: он может и рукой в экран постучать, и заговорить, и просто приветливо помахать.

Реализуя реакцию персонажа на действия пользователя, лучше всего заранее продумать синонимический ряд ответов, как в плане анимации, так и в плане речи (если она используется). Повторение одних и тех же слов или проигрывание одной и тоже же анимации в ответ на запрос пользователя создадут неприятное впечатление, а также вызовут эффект раздражения. Ведь мы — люди — никогда не говорим одинаковыми фразами (слово в слово) даже в идентичных ситуациях, и над этим тоже стоит задуматься, тем более что MS Agent уже обладает некоторыми вспомогательными функциями для воплощения данной идеи. Например, метод, запускающий чтение текста персонажем, предусматривает возможность случайного выбора фразы для воспроизведения из синонимического ряда (подробнее мы рассмотрим это в последующих разделах).

Рекомендации социального характера

Основная задача персонажа MS Agent, как элемента интерфейса, — обеспечить социальный контакт с пользователем. Поэтому будет неплохо если вы, не прибегая к искусственному интеллекту, наделите ассистента некой индивидуальностью через стиль его поведения, движения, используемые реплики. Это приносит хорошие результаты особенно тогда, когда стиль, созданный вами, гармонирует с внешностью персонажа. Я полагаю, никому не придет в голову обучить старичка Мерлина (Мага-звездочета) молодежным выражениям и раскованному поведению. Важно также, чтобы ваш персонаж придерживался простейших правил этикета, иначе его использование будет иметь лишь отрицательный эффект.

Человек не любит аморфных и неоднозначных собеседников, так что вам следует четко определить индивидуальность персонажа. Если его роль заключается в том, чтобы помочь пользователю справиться с возникающими у него вопросами, пусть демонстрирует покорность. Однако если роль агента — быть гидом в вашем приложении и направлять действия пользователя, то лучше использовать доминирующее поведение. Тем не менее, все же не забывайте о компетентности агента, ведь вряд ли вы сможете программно узнать причину поведения пользователя: почему он, например, воспользовался инструментом по своему выбору, а не тем, каким советовал ему воспользоваться персонаж.

Исследования показали, что люди предпочитают общаться с людьми, похожими на них самих, поэтому позаботьтесь о том, чтобы пользователь имел возможность выбирать персонажи из возможного списка.

Рекомендации по синтезу речи

Речевой вывод информации — естественная форма связи приложения с пользователем, но следует учитывать, что у последнего может не оказаться необходимых компонентов. Так что советую не забывать дублировать выводимые сообщения в других элементах пользовательского интерфейса.

При чтении персонаж выводит произносимый текст в "воздушном шарике". Это очень эффектно, однако пользователь может изменить настройки и отключить вывод "воздушного" текста. Не забывайте программно контролировать данный параметр, чтобы пользователь вовремя получал необходимую информацию от приложения.

Не увлекайтесь речевым выводом, т. к. это далеко не самый быстрый способ подачи информации. Естественно, без речевого вывода не обойтись, если он является целью вашего приложения, как, например, при реализации чтения электронных книг, но в этом случае необходимо позаботиться о разбиении текста на части.

Используйте простые (устные) выражения для лучшего понимания смысла при подаче информационных сообщений, т. к. они будут произноситься голосом, а на слух воспринимать запутанные выражения трудно. По той же причине речи персонажа незачем изобиловать слишком вежливыми выражениями — информация, особенно важная, должна быть кратка и понятна. Хорошо также будет, если некоторые сообщения будут сформулированы от первого лица — это еще более приблизит персонаж к пользователю.

Отладьте чтение статических выражений в программе. То есть добейтесь того, чтобы все выражения, которые не будут изменяться во время выполнения программы, читались правильно (имеются в виду ударения в словах, логические ударения и т. п.). Для этого существуют специальные инструменты, которые мы рассмотрим позже *(см. разд. 2.8)*.

Советы по распознаванию речи

При использовании распознавания речи не перекладывайте слишком много команд на эту функцию. Устойчиво работает распознавание англоязычной речи, поэтому все команды придется подавать только на английском языке (естественно, можно подавать их и на русском, если вы найдете соответствующий движок). Команды должны быть простыми и запоминающимися. Оцените, какие действия быстрее будет запустить обычными средствами.

Не создавайте похожих команд — хотя их список можно открыть в отдельном окне, идея распознавания речевых команд заключается в том, чтобы ускорить их запуск. Конечно, проще сказать одно слово, чтобы вызвать выполнение какого-либо действия, чем долго тянуться мышью до элемента управления, но тем не менее, команда (ее речевая формулировка) должна

быть запоминающейся, чтобы пользователь не тратил много времени для ее заучивания.

Вы, вероятно, понимаете, что распознавание голосовых команд производится с большими ошибками. Не рассчитывайте особенно на голосовой ввод и используйте, по возможности, формулировки средней длины. Короткие команды типа "Up" обречены на неудачу, любой шум или шипение могут быть приняты за реальный голос пользователя. При распознавании слишком длинных формулировок алгоритмы также могут "запутаться".

Придумывайте выражения для команд, не созвучные на слух. Например, команды "Speak time!" и "Speak name!" вряд ли будет возможно стопроцентно правильно отличить друг от друга.

Выводы

Итак, кратко сформулируем правила, которых следует придерживаться при создании анимированного персонажа.

- Общие:
  - определите роль агента;
  - дайте пользователям возможность выбрать, использовать им персонаж или нет;
  - обеспечьте обратную связь с ассистентом;
  - рационально применяйте анимацию персонажа;
  - подготовьте синонимические ответы пользователю.
- Социальные:
  - создайте яркую индивидуальность из персонажа;
  - соблюдайте этикет при построении поведения агента.
- Синтеза речи:
  - дублируйте сообщения другими способами;
  - используйте эффективные (краткие) формулировки;
  - применяйте инструменты коррекции речи.
- Распознавания речи:
  - создавайте простые и удобные речевые команды (их не должно быть много);
  - речевые команды не должны быть похожи друг на друга;
  - найдите оптимальную длину формулировки речевой команды.

2.3. Объектная модель Microsoft Agent Object

Наконец-то мы приступаем к программированию. Как уже упоминалось, пользоваться методами интерфейсов сервера Microsoft Agent напрямую не очень удобно, т. к. в изобилии интерфейсов и их методов легко потеряться, особенно новичку.

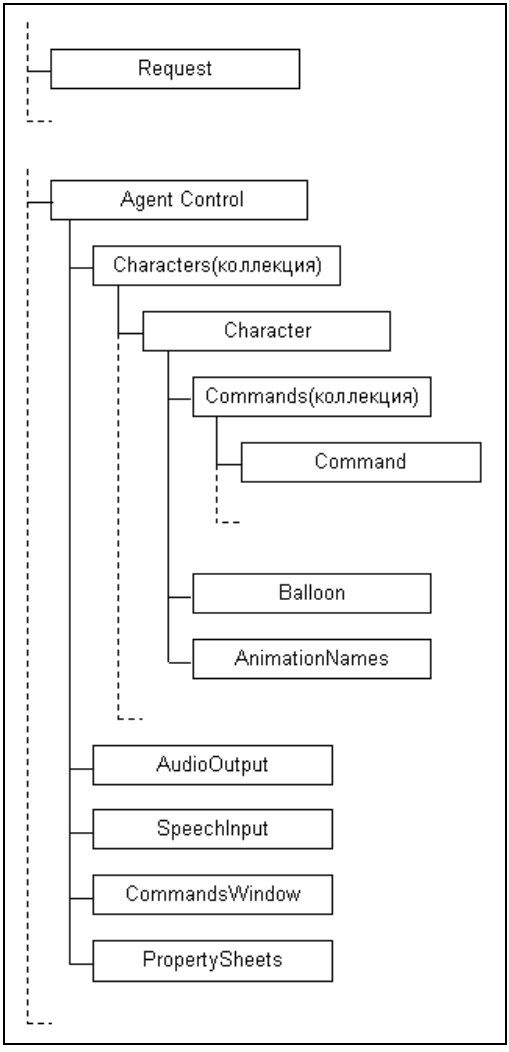


Рис. 2.13. Схема объектной модели Microsoft Agent

Для решения этой проблемы следует применять специальный элемент управления (Agent Control), реализованный как ActiveX-компонент. Ясно, что этот элемент реализует несколько иные интерфейсы, специально созданные для него. Однако его класс имеет очень удобное построение, в нем четко прослеживается логическая структура.

Предлагаемая Microsoft модель агента разделена на несколько элементов, называемых объектами. Однако не следует понимать их как объекты какого-либо класса, т. к. это не указывается явно в документации к Microsoft Agent. Нужно привыкнуть представлять себе все объекты как наборы интерфейсов, реализуемых где-то внутри сервера и элемента управления. Естественно, что для конечных разработчиков недоступна информация по части организации сервера. В любом случае нас не должно волновать, как там все это "крутится". Главное — знать, что за каждым обозначенным объектом скрывается набор интерфейсов, реализованных в OLE Automation-сервере.

Рассмотрим объектную модель элемента управления MS Agent. Общая схема компонентов представлена на рис. 2.13. Проанализируем ее коротко, а затем, при рассмотрении вопросов программирования, коснемся подробно каждого элемента.

Объект *Agent Control*

Этот объект обеспечивает доступ и связь с другими объектами сервера Microsoft Agent, а также позволяет обращаться к его свойствам и методам. Это не сам персонаж и не сервер автоматизации, а только элемент управления, однако именно через него мы можем получить доступ и к самому персонажу (или персонажам), и к настройкам, затрагивающим всех ассистентов, к речевому выводу и вводу и т. д, чтобы не обращаться напрямую к серверу MS Agent. Кроме того, Agent Control имеет некоторые дополнительные свойства.

`Agent Control` включает в себя специальную структуру — коллекцию персонажей (Characters Object), которая хранит все загруженные персонажи. Через нее можно обратиться к конкретному агенту по его идентификатору.

Каждый персонаж в коллекции представляет собой объект Character, который обладает набором команд (Commands), определенным стилем выноски (воздушнго шарика) (Balloon), причем имеется в виду, что для каждого ассистента можно по-своему настроить стиль воздушного шарика, и, конечно, списком возможных анимаций — AnimationNames.

Каждая анимация в коллекции имеет свое строковое имя, по которому ее можно вызвать. Существуют некоторые стандартные названия анимаций, которые проигрываются персонажем самостоятельно в определенных ситуациях. Например, анимация Show — при появлении агента, Hide — при скрытии и т. д. Разработчикам своих собственных персонажей необходимо подумать над наименованием каждой анимации, создать стандартные ани-

мации, чтобы персонаж мог устойчиво функционировать в среднестатистическом приложении, использующем MS Agent, а программисту, пишущему приложение, — позаботиться об обработке исключения, если запускаемая анимация с указанным именем отсутствует.

Специальные объекты для речевого ввода и вывода (SpeechInput и AudioOutput соответственно) позволяют управлять некоторыми параметрами ввода речевых команд и вывода аудиоданных (речи, различных эффектов и др.).

Для получения свойств окна речевых команд и изменения некоторых из них служит специальный объект CommandsWindow, а объект PropertySheet позволяет контролировать параметры диалогового окна **Дополнительные параметры персонажей** (Advanced Character Options).

Объект *Request*

Этот вспомогательный объект не относится к элементу управления MS Agent, т. е. не является его дочерним элементом. Объект Request необходим для получения некоторой вспомогательной информации о действиях персонажа.

Мы уже знаем, что собой представляет MS Agent с точки зрения пользователя, и вы, конечно, понимаете, что некоторые действия персонажа, такие как чтение и проигрывание анимации, происходят асинхронно. Сервер ставит в очередь потока персонажа запросы, предназначенные для него, а код приложения-клиента продолжает выполняться со следующей команды. Используя один из таких асинхронных методов, элемент управления создает и возвращает объект `Request` для каждого запроса, а сервер MS Agent автоматически уведомляет клиента через данный объект о том, в каком состоянии находится запрос. Если программист создает переменную для хранения ссылки на объект типа `Request`, то в программном коде возможно будет устанавливать состояние запроса: например, находится ли он в процессе выполнения, стоит ли в очереди, закончился неудачей или присутствует в других состояниях.

2.4. Начало программирования персонажей

Перед подробным рассмотрением интерфейсов, методов и свойств MS Agent я продемонстрирую простейшие приложения с использованием персонажа. Надеюсь, это облегчит понимание дальнейшего описания.

Рассмотрим сначала прямое обращение к серверу Agent Server, чтобы понять механизм работы MS Agent как сервера автоматизации, т. е. без использования специального элемента управления Agent Control. Затем рассмотрим бескомпонентное (без визуального ActiveX) программирование с использованием позднего связывания с Agent Control вместе с VCL (Visual Compo-

nent Library, библиотека визуальных компонентов) и в WinAPI, и далее обратим внимание на отличия подхода раннего связывания с использованием библиотеки типов от позднего. И наконец, перейдем к визуальному программированию с Agent Control (компонентом ActiveX) для создания приложений с анимированным персонажем Microsoft Agent.

Разные варианты подходов применяются в различных ситуациях. Хоть позднее связывание считается более медленным способом, однако иногда без него не обойтись. Тем более что в случае с MS Agent скорость доступа при позднем и раннем связывании практически одинакова, т. к. клиент и сервер находятся в разных потоках.

Посмотреть использование сервера MS Agent напрямую также будет полезно, т. к. позволит понять внутреннюю организацию его интерфейсов.

Позднее связывание может использоваться при отсутствии библиотеки типов или при написании небольшого приложения (не исключено, что на WinAPI). Раннее связывание рациональнее использовать при создании приложений, применяющих большое количество функций MS Agent, для того чтобы обнаружить все синтаксические ошибки еще на этапе компиляции.

Программирование с применением визуального ActiveX представляет собой еще один способ работы с MS Agent. Вместе с тем визуальный компонент не слишком облегчает написание программ по сравнению с тем, насколько раннее связывание с Agent Control упрощает программирование при переходе к нему от позднего. Знание этого способа может пригодиться в Delphi, учитывая привычность обращения с компонентами, а также удобное построение класса `TAgent`. Тем не менее, облегчение программирования компонентным способом может дорого обойтись в том случае, если у пользователя не установлен Microsoft Agent. Легче обработать нулевое значение переменной типа `Variant` или интерфейсного типа, чем ошибку создания компонента ActiveX на этапе выполнения кода. Естественно, это соображение — не причина отказываться от некоторых удобств, тем более что по последним данным Windows 2000 и Windows XP, где уже установлен MS Agent, используется на большей части компьютеров мира. Ну, а остальных пользователей вы уж предупредите о том, что ваша программа требует установки Microsoft Agent.

Для начала определим порядок действий, которые нужно проделать для того, чтобы начать работать с одним персонажем:

1. Подключиться к серверу Microsoft Agent.
2. Загрузить персонаж в коллекцию.
3. При необходимости показать персонаж.

В зависимости от подхода мы по-разному будем подключаться к серверу MS Agent. В случае прямого обращения будем запрашивать создание экземпляра объекта у библиотеки COM. При подключении с использованием

Agent Control нам поможет сам элемент управления. В случае с визуальным компонентом ActiveX он сам будет подключаться к серверу при запуске приложения.

Способ загрузки персонажей в коллекцию также будет зависеть от подхода. В первом варианте придется пользоваться методами интерфейсов Agent Server, в случае с Agent Control применять интерфейсы элемента управления, а также методы, реализованные в классе `TAgent`.

Загруженный персонаж можно использовать по его прямому назначению, и далее разработчик свободен в своих действиях. Тем не менее, поскольку в большинстве случаев персонаж отображают на экране, мы будем поступать с нашими минимальными приложениями так же.

Стоит заметить, что во всех примерах данного раздела преимущественно используется персонаж *Merlin* (Мерлин, Маг, Звездочет), а в нескольких — пресонажи *Genie* (Джинн) и *Peedy* (Попугай, Кеша). Вам, следовательно, необходимо установить эти персонажи, или — при наличии желания применять другие персонажи для рассмотрения работы демонстрационных примеров — в кодах придется изменять имена файлов загружаемых агентов.

Прямое обращение к Agent Server

Предполагаю, что некоторым начинающим программистам не терпится увидеть, как на экране появится персонаж под их командованием. Приведу небольшой код, который демонстрирует простое появление персонажа и больше ничего.

Познакомимся с кодом на примере использования сервера MS Agent. Попробуем обратиться не к элементу управления, а напрямую к Agent Server, пользуясь его интерфейсами и методами.

Как вы понимаете, для использования сервера в раннем связывании (а мы рассмотрим только его) требуется подключение библиотеки типов, т. е. придется присоединять к проекту заголовочный файл с объявлением всех интерфейсов, процедур, функций, констант и т. д. Необходимый модуль для разных версий Delphi располагается на носителе. Новички спросят: что же делать, если у нас нет этого файла? В этом случае надо импортировать библиотеку типов из сервера MS Agent. В среде Delphi для этого требуется выбрать пункт **Project | Import Type Library** (Проект | Импортировать библиотеку типов), в появившемся диалоговом окне (рис. 2.14), подобном окну импорта **ActiveX Control**, следует выбрать необходимый элемент, в данном случае **Microsoft Agent Server v2.0**, и нажать кнопку **Create Unit** (Создать модуль). В результате по умолчанию в папке $(Delphi)\Import\, если вы ее не меняли, создастся файл AgentServerObjects_TLB.pas, где $(Delphi) — каталог с установленной средой программирования. Вот мы и получили библиотеку

типов. Замечу, что на прилагаемом компакт-диске в примерах даются переименованные файлы, без суффикса `_TLB`, чтобы при подключении не путать их с файлом, созданным средой.

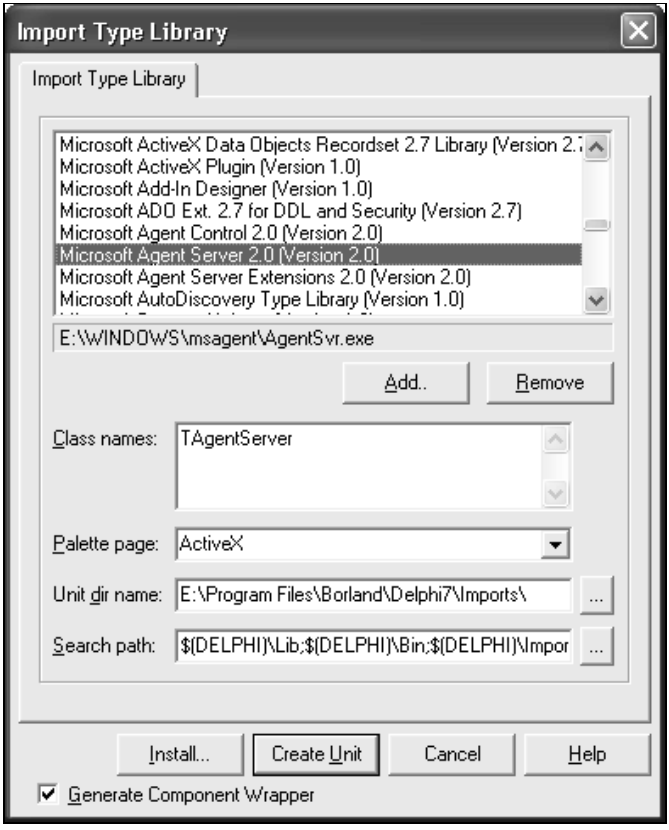


Рис. 2.14. Диалоговое окно импорта библиотеки типов

Приступая к программированию, первым делом подключим необходимый модуль, а затем в процедуре обработки события создания главной формы приложения пропишем загрузку персонажа. Сначала через процедуру `CoCreateInstance` обратимся к библиотеке COM для получения первого интерфейса объекта — `IAgentEx`. (Через интерфейс `IAgentEx` можно загружать персонажи, обращаться к Speech Input/Output и т. п.) Затем загрузим персонаж и покажем его на экране. Код модуля приведен в листинге 2.1, это проект каталога 01_AgentServerEx на компакт-диске. Окно приложения с персонажем показано на рис. 2.15.

Листинг 2.1. Модуль главной формы приложения обращения к Agent Server

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, ActiveX, AgentServerObjects;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  Agent: IAgentEx;
  Chars: IAgentCharacterEx;
  lCharID: integer;
  lReqID: integer;

implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  CoCreateInstance(CLASS_AgentServer, nil, CLSCTX_SERVER, IID_IAgentEx,
Agent);
  Agent.Load('merlin.acs', lCharID, lReqID);
  Agent.GetCharacterEx(lCharID, Chars);
  Chars.Show(0, lReqID);
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  Agent.Unload(lCharID);
end;

end.
```

Ну как, запустили проект? Могу поспорить, что многие читатели сейчас подпрыгнули от восторга. Наконец мы написали свое первое приложение с MS Agent, да к тому же оно оказалось очень простым! Правда, персонаж просто появляется на экране и ничего не делает, озирается вокруг, и только. Однако это и есть минимальное приложение с MS Agent.

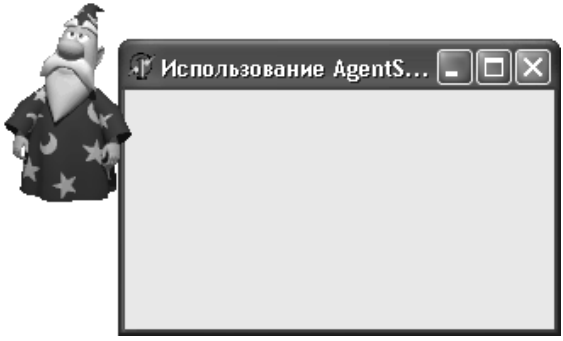


Рис. 2.15. Окно проекта каталога 01_AgentServerEx

Разберем процедуру `FormCreate` построчно. Прежде всего получаем указатель на интерфейс `IAgentEx`. Данный интерфейс позволяет нам загружать и выгружать персонажи, регистрировать объекты, реализующие реакцию на некоторые события MS Agent (notification sink). Интерфейс `IAgentEx` образован от `IAgent`, т. к. к интерфейсу добавлены новые методы, такие как `ShowDefaultCharacterProperties` и `GetVersion`. Из названий видно, что первый метод отображает диалоговое окно со свойствами персонажа, а второй — версию Agent Server. В данном примере эти методы не используются, поэтому с таким же успехом можно было применить интерфейс `IAgent`. Тем не менее, пример не будет успешно откомпилирован, если стереть пару символов в названии типа соответствующей переменной. Это происходит потому, что метод `GetCharacter` тоже обновлен, а вернее — в интерфейс добавлен расширенный метод получения интерфейса персонажа — `GetCharacterEx`. Скоро мы рассмотрим это подробнее, но прежде обратим внимание на следующую (вторую) строку процедуры `TForm1.FormCreate`, где вызывается метод `Load` интерфейса `IAgentEx`.

В методе `Load` определены три параметра. Один входной — имя файла персонажа, и два выходных — идентификатора персонажа (character ID) и идентификатор, хранящий состояния запроса (request ID). Идентификатор персонажа создается автоматически и нужен он для того, чтобы при одновременном использовании нескольких ассистентов обращаться к каждому по-отдельности. Идентификатор уникален для каждого сеанса загрузки, следовательно может использоваться только внутри приложения для отличия одного персонажа от другого, поскольку даже если иное приложение или вторая копия той же программы обращается к уже используемому ассистенту, идентификатор будет другим. Третьим параметром метод `Load` возвращает индентификатор запроса.

Итак, переходим к третьей строке процедуры `TForm1.FormCreate`. Объект Agent Server мы уже создали, персонаж в коллекцию загрузили, осталось показать его на экране. Однако показать не сразу: прежде чем отобразить персонаж методом `Show`, необходимо получить интерфейс персонажа `IAgentCharacter` или `IAgentCharacterEx` по его идентификатору. Разница между двумя интерфейсами в основном заключается в том, что во второй, расширенный, интерфейс добавлены дополнительные методы для работы с речевым выводом и вводом. Получение интерфейса персонажа по его идентификатору (переменная `lCharID`) в данном случае нужно производить через метод `GetCharacterEx`, т. к. используется интерфейс `IAgentEx`, а данный метод возвращает указатель на `IAgentCharacterEx`.

Получив указатель на интерфейс определенного персонажа, можно делать с ним все, что угодно, поэтому отобразим его на экране методом `Show` и на этом закончим наш пример. Как видите, состояние запроса также возвращается в программу через выходной параметр. Еще один (первый) параметр метода показывает, нужно ли пропускать проигрывание анимации Show при показе персонажа. Если его значение равно 0 (т. е. `False`), то анимация не пропускается, иначе (в случае 1, или `True`) анимация пропускается и персонаж появляется без задержки.

Замечание

Вы можете загрузить много персонажей, однако невозможно загрузить несколько раз персонаж с одним и тем же GUID. Каждый ассистент имеет свой собственный GUID, который можно посмотреть в окне свойств файла персонажа на вкладке **Версия** (рис. 2.16). Иными словами, при попытке загрузить новую копию уже загруженного персонажа (вами или другим приложением) ровным счетом ничего не произойдет. В программе вы получите указатель на уже запущенный персонаж, и управлять будете им.

Замечание

Предполагаю, что может возникнуть вопрос: почему в виде идентификатора используется GUID? Естественно, — благодаря его алгоритмам генерации, га-

рантирующим, что он не повторится ни на одном компьютере мира при создании нового сочетания. Учитывая требование о том, что Microsoft Agent не должен загружать два одинаковых персонажа (иначе как пользователь будет отличать, например, один загруженный персонаж Genie от его второй копии?), более подходящего типа для идентификатора не найти.

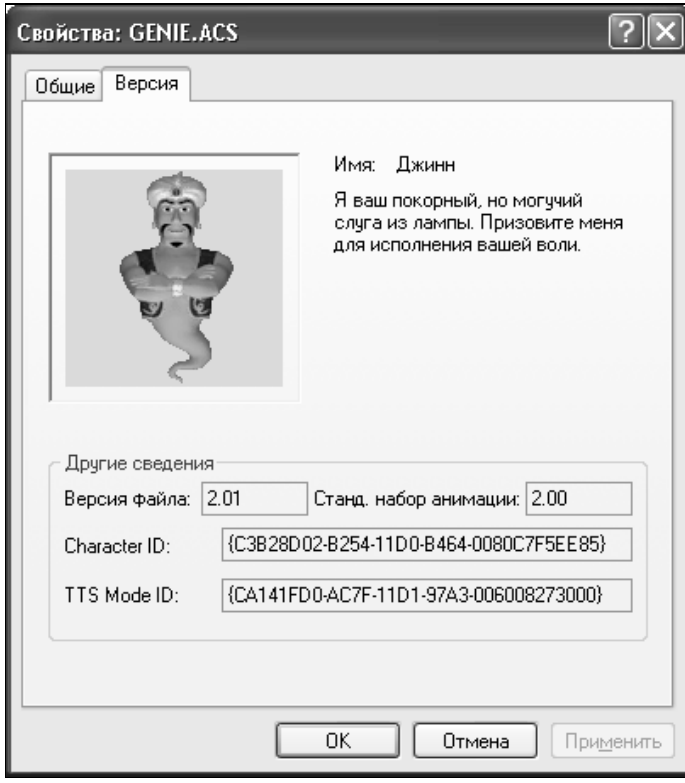


Рис. 2.16. Вкладка **Версия** окна свойств файла ACS (в данном случае — genie.acs)

При уничтожении формы желательно также не забыть прописать выгрузку персонажа по его идентификатору (Character ID, переменная `lCharID`) методом `Unload` интерфейса `IAgentEx`. Персонаж автоматически выгружается, когда отпадает необходимость в нем, т. е. если вы забыли в приложении программно выгрузить задействованные персонажи, то ошибки не возникнет — они сами исчезнут и очистят память, как только отпадет необходимость в последней ссылке на них. Тем не менее, особенно в этом примере с Agent Server может наблюдаться некоторая задержка при исчезновении персонажа после закрытия приложения, если не создать выгрузку его вручную.

Это связано с тем, что COM-сервер не сразу проверяет количество ссылок на объект. Поэтому существует специальный метод `Unload`, который выгружает персонажи. Этот метод используется, например, для того, чтобы сменить персонажа. Допустим, вы используете в приложении один персонаж под каким-либо идентификатором, а при команде пользователя необходимо сменить текущий персонаж на новый. В этом случае требуется предварительно выгрузить настоящий персонаж методом `Unload`, а затем загрузить новый методом `Load`. Применение `Unload` не означает, что персонаж удален с экрана и за ним очищена память (не забывайте — текущий персонаж может использовать другое приложение). Зато указатель на персонаж будет однозначно удален из коллекции персонажей, и к нему невозможно будет обратиться по прежнему идентификатору из текущей программы, даже если его использует стороннее приложение.

Проиллюстрируем сказанное. Запустите одну копию нашего первого приложения. На экране появится персонаж. Затем запустите вторую копию приложения и убедитесь, что еще один персонаж уже не отображается, т. к. такой ассистент уже загружен. Если вы скроете персонаж после запуска первой копии, то при загрузке второй он вновь "выскочит", потому что выполнится только метод `Show`. Теперь попробуем завершить выполнение первой копии программы. Персонаж не исчез, хотя при уничтожении формы явно прописана его выгрузка. Только при закрытии второй копии программы персонаж скроется.

Можно проверить подсчет ссылок на персонаж. Создадим одно приложение без выгрузки персонажа по завершении, и одно — с выгрузкой. Проделав аналогичные действия, мы заметим, что после уничтожения первого и второго (последнего, использующего персонаж) приложения персонаж автоматически исчезает. Следовательно, механизм подсчета ссылок на персонаж работает верно и без использования метода `Unload`. Однако он не обязательно сработает сразу — агент, возможно, будет долго одиноко висеть на экране до тех пор, пока сервер не "сообразит", что данный персонаж никто не использует.

Замечание

Хочется отметить, что описанная задержка не имеет места при использовании Agent Control. При использовании элемента управления персонажи всегда четко выгружаются, как только их перестает использовать какое-либо приложение.

Посмотрим теперь, как заставить правильно работать приложения с интерфейсом `IAgent`, а не `IAgentEx` Возможно применять в этом примере и интерфейс `IAgent`, однако, как мы видели, он неудачно компилируется. Все потому, что в нем отсутсвует метод `GetCharacterEx`, зато имеется другой — `GetCharacter`. Данный метод возвращает не указатель на интерфейс персо-

нажа, а указатель на IDispatch. С использованием интерфейса IAgent код приложения увеличится на одну строчку вследствие того, что нужно будет запросить через метод QueryInterface указатель на необходимый нам IAgentCharacter (см. листинг 2.2).

Листинг 2.2. Обработка события создания формы и загрузка персонажа с использованием интерфейса IAgent (пример подкаталога 02_AgentServer)

```
var
  Form1: TForm1;
  Agent: IAgent;
  Chars: IDispatch;
  Ch: IAgentCharacter;
  lCharID: integer;
  lReqID: integer;

...

procedure TForm1.FormCreate(Sender: TObject);
begin
  CoCreateInstance(CLASS_AgentServer, nil, CLSCTX_SERVER, IID_IAgent,
Agent);
  Agent.Load('merlin.acs', lCharID, lReqID);
  Agent.GetCharacter(lCharID, Chars);
  Chars.QueryInterface(IID_IAgentCharacter, Ch);
  Ch.Show(0, lReqID);
end;
```

Как видим, разницы нет — тогда зачем писать на строку больше?

Взаимодействовать с сервером напрямую не очень приятно, тем более зная, что создан специальный элемент управления для более удобного программирования. Так что перейдем к рассмотрению элемента управления Agent Control.

Раннее и позднее связывание с элементом управления Agent Control

Работать со специальным элементом управления Agent Control гораздо удобнее. Приведу пример приложения с его использованием (листинг 2.3). Используя данный компонент, надо понимать, что в этом случае мы будем

работать с интерфейсами несколько иными, чем в случае Agent Server. В приведенном коде использовано позднее связывание с Agent Control, т. к. это наиболее простой способ взаимодействия с MS Agent, не требующий никаких библиотек типов.

Листинг 2.3. Простейшее приложение с MS Agent (пример из каталога 03_FirstShow)

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, ComObj;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  Agent: Variant;

implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  Agent:=CreateOleObject('Agent.Control.2');
  Agent.Connected:=1;
  Agent.Characters.Load('merlin','merlin.acs');
  Agent.Characters.Character('merlin').Show(0);
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  Agent.Characters.UnLoad('merlin');
end;

end.
```

Разберем построчно приведенный код. Во-первых, обратите внимание на переменную `Agent` типа `Variant`, которая ассоциируется с элементом управления Microsoft Agent (это делается в первой строке процедуры обработки события создания главной формы приложения TForm1.FormCreate). Строковая константа `Agent.Control.2` соответствует информативному имени элемента управления.

Интересно быть в курсе того, как я это узнал и как любой из читателей сможет увидеть это. Один из способов узнать имя элемента управления заключается в изучении ресурсов библиотеки, в которой располагается элемент управления Agent Control (ActiveX), в данном случае Agentctl.dll. Воспользуйтесь любым просмотрщиком ресурсов вроде Restorator (от Florian Bomers, **http://www.bome.com/Restorator/**) или Resource Hacker (от Angus Johnson, **http://rpi.net.au/~ajohnson/resourcehacker**). Откройте вышеназванную библиотеку, отыщите секцию `Registry` — в ней должен находиться ресурс с именем 101. Внимательно изучите его. Подобные секции с идентичными ресурсами присутствуют в некоторых библиотеках, содержащих OLE Automation, однако это не правило, поскольку все зависит от того, какова структура библиотеки и как она была написана. Рассмотрим указанный ресурс (рис. 2.17). Из его содержания понятно, какие ключи прописываются в секцию `HKCR` (`HKEY_CLASSES_ROOT`). Раздел `Agent.Control.1` содержит перекрестную ссылку на Microsoft Agent v1.5, `Agent.Control.2` — соответственно, на версию 2.0.

Если бы нам был известен CLSID автоматизации, то можно было бы найти соответствующую запись в реестре прямым поиском, чтобы узнать имя раздела и тем самым определить информативное имя сервера OLE Automation.

После того как объект создан, необходимо подключиться к серверу Microsoft Agent, для чего свойству `Connected` присваивается значение 1, что равнозначно `True`. Соединение установлено, следовательно можно пользоваться всеми возможностями и функциями, предоставляемыми элементом управления. Первым делом загружаем персонаж — он главный объект нашего внимания.

Метод `Load` коллекции персонажей `Characters` загружает в нее персонаж по имени файла. Если файл персонажа располагается в папке \%WinDir%\MS Agent\Chars, то можно указать лишь имя файла; в других случаях при-

дется указывать полный путь к персонажу, даже если его файл находится в каталоге с программой.

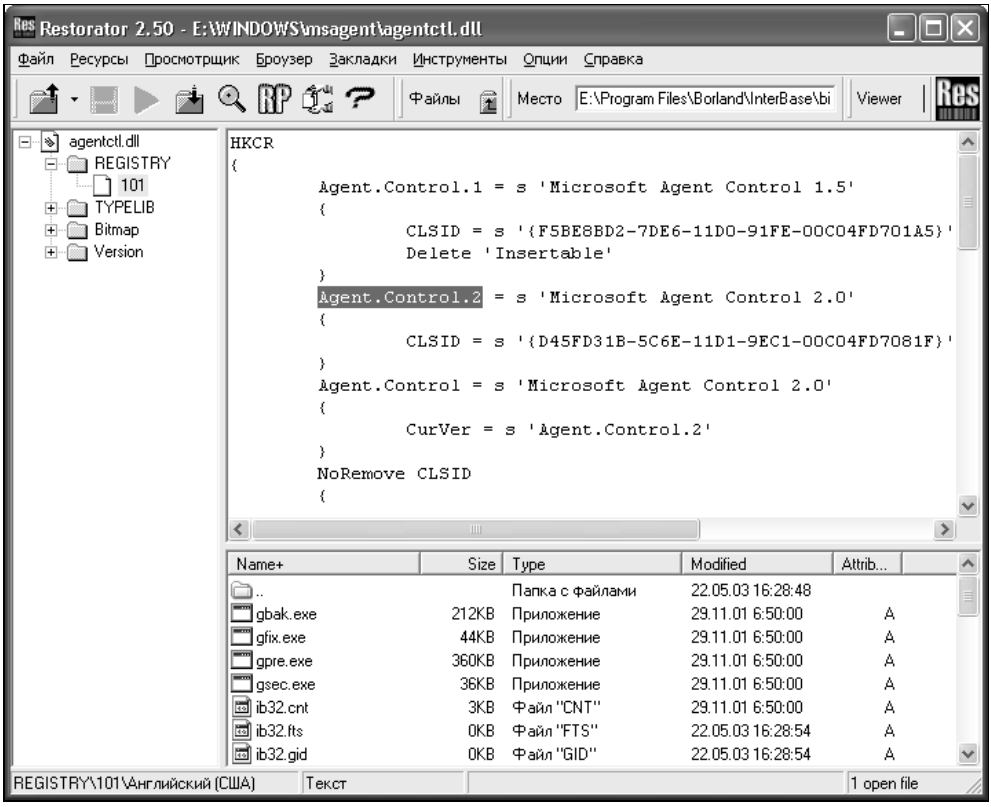


Рис. 2.17. Главное окно программы Restorator при просомтре ресурсов библиотеки Agentctl.dll

Метод `Load` имеет два параметра. По виду второго параметра вы догадались о том, что с ним должно передаваться имя файла (и это действительно так), а вот первый параметр не так красноречив, хотя также немаловажен. Он является строковым идентификатором персонажа в нашем приложении. Так, например, при работе с несколькими персонажами одновременно необходимо как-то обращаться к ним, находить их в коллекции персонажей элемента управления. При использовании Agent Control идентификатор создается программистом. С одной стороны, это удобно, т. к. мы придумываем информативное имя, а не просто получаем какое-то число, как в случае прямого обращения к серверу MS Aagent.

Замечание

При обращении к персонажу через его идентификатор не имеет значения регистр символов. То есть идентификаторы `Merlin` и `MERlin` будут указывать на один и тот же персонаж.

Метод `Show` является функцией и возвращает состояние запроса не так, как это делает метод `Show` интерфейса `IAgentEx` или `IAgent` в примере с Agent Server.

В данном простейшем примере мы не используем информацию о состоянии запроса (как, впрочем, и в предыдущих примерах), однако здесь необязательно сохранять ее в переменной.

Итак, осталась не прокомментированной последняя строка процедуры `FormCreate`. Тем не менее очевидно, что именно она показывает персонаж на экране, т. к. мы обращаемся к конкретному персонажу коллекции по имени и применяем к нему метод `Show`. В данном случае не нужно специально получать указатель на интерфейс персонажа, как при работе с Agent Server — он неформально возвращается при обращении к элементу коллекции персонажей, т. е. к конкретному ассистенту.

Существует специальный метод у объекта `Characters` (коллекции персонажей) элемента управления, а также `IAgent` — метод `Unload`. С его помощью выгружаются персонажи, которые больше не нужны вашей программе. Данным методом можно и не пользоваться, поскольку ненужные персонажи сами собой будут удалены из памяти. Кроме того, при использовании Agent Control без применения `Unload` задержки при уничтожении персонажа не возникает, в отличие от ситуации в примере с Agent Server.

Замечание

По всей видимости, это связано с тем, что сам элемент управления при уничтожении проверяет количество ссылок на невыгруженных персонажей.

Посмотрим, что изменится при программировании на WinAPI. Рассмотрим код простейшей программы (см. листинг 2.4).

Листинг 2.4. Простейшая программа с MS Agent на WinAPI (некоторые комментарии опущены)

```
program AgentAPI;
uses
 windows, messages, ComObj, ActiveX;
var
 wc: TWndClassEx; //Переменная шаблона класса окна
 MainWnd: HWND;  //Описатель главного окна
```

```
 Mesg: TMsg;      //Переменная для цикла сбора сообщений
 szClassName: PAnsiChar; //Переменная для хранения имени класса
 Agent: Variant;
 xPos,yPos,nWidth,nHeight : Integer;

{ Оконная процедура главного окна}
function WindowProc(wnd: HWND; Msg: Integer; Wparam: WPARAM; Lparam:
LPARAM):LResult; stdcall;
begin
  case Msg of
   WM_SHOWWINDOW://Сообщение, посылаемое при показе окна
     begin
      CoInitialize(nil);
      Agent:=CreateOleObject('Agent.Control.2');
      Agent.Connected:=1;
      Agent.Characters.Load('merlin','merlin.acs');
      Agent.Characters.Character('merlin').Show(0);
      Result:=0;
     end;
   WM_DESTROY://Сообщение, посылаемое при уничтожении окна
     begin
      Agent.Characters.UnLoad('merlin');
      PostQuitMessage(0);
      Result:=0;
      exit;
     end;
   else
      Result:=DefWindowProc(wnd, Msg, Wparam, Lparam);
  end;
end;

begin
 szClassName:='AgentInAPI';//Имя класса
{ Заполнение шаблона класса окна}
 wc.cbSize:=SizeOf(wc);
 wc.style:=CS_HREDRAW or CS_VREDRAW;
 wc.lpfnWndProc:=@WindowProc;
 wc.cbClsExtra:=0;
```

```
 wc.cbWndExtra:=0;
 wc.hInstance:=HInstance;
 wc.hIcon:=LoadIcon(0,IDI_APPLICATION);//описатель пиктограммы окна
 wc.hCursor:=LoadCursor(0,IDC_ARROW);//Курсор(обычный)
 wc.hbrBackground:=COLOR_BTNFACE+1;//Фон
 wc.lpszMenuName:=nil;
 wc.lpszClassName:=szClassName;//Присваеваем имя класса

 RegisterClassEx(wc);//Регистрация нового класса в системе

 xPos:=100;
 yPos:=150;
 nWidth:=400;
 nHeight:=250;
 {Создание главного окна}
 MainWnd:=CreateWindowEx(
   0,                  //флаги расширенных стилей
   szClassName,        //имя класса окна, данное при заполнении wc
   'MS Agent из API',   //заголовок окна
   WS_OVERLAPPEDWINDOW,//флаги стилей окна
   xPos,               //горизонтальная позиция окна
   yPos,               //вертикальная позиция окна
   nWidth,             //ширина окна
   nHeight,            //высота окна
   0,                  //описатель родительского окна или окна-владельца
   0,                  //описатель меню окна
   Hinstance,          //описатель приложения
   nil                 //Адрес данных для создания окна
 );
 ShowWindow(MainWnd,CmdShow);//Отображаем окно
 //Цикл обработки сообщений
 while GetMessage(Mesg,0,0,0) do
  begin
   TranslateMessage(Mesg);
   DispatchMessage(Mesg);
  end;
end.
```

Как видите, код, подключающий персонаж, не особенно отличается от кода предыдущего примера, однако нужно не забывать перед созданием объекта инициализировать COM и поддержку многопоточной памяти. Это делает процедура `CoInitialize`.

Ознакомившись с подходом позднего связывания, посмотрим на аналогичный код, написанный с использованием подхода раннего связывания (см. листинг 2.5).

Разница почувствуется лишь в одной первой строке, где вместо обращения к процедуре `CreateOleObject` необходимо обратиться к библиотеке COM (чтобы создать нужный объект) и в объявлении типов переменных. Здесь требуется подключить дополнительный модуль с библиотекой типов — его заголовочный файл AgentObjects.pas содержится на компакт-диске для разных версий Delphi. Тот же файл создается при импорте ActiveX в Delphi, правда, называется он AgentObjects_TLB.pas. Я намеренно его переименовал, чтобы не путаться в проектах.

Листинг 2.5. Пример раннего связывания с Agent Control (каталог 05_AgentEarly)

```
unit Unit1;


interface


uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, ActiveX, AgentObjects;


type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;


var
  Form1: TForm1;
  Agent: IAgentCtlEx;
```

```
implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  CoCreateInstance(CLASS_Agent, nil, CLSCTX_ALL, IID_IAgentCtlEx, Agent);
  Agent.Connected:=true;
  Agent.Characters.Load('merlin','merlin.acs');
  Agent.Characters.Character('merlin').Show(0);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Agent.Characters.UnLoad('merlin');
end;

end.
```

Программирование MS Agent с ActiveX

Ранее я описывал установку элемента управления Microsoft Agent Control v2.0 *(см. разд. 2.1, подразд. "Установка компонентов и импорт ActiveX"),* а сейчас мы рассмотрим пример минимального приложения MS Agent с применением компонента ActiveX. Это тот же самый Agent Control в своем визуальном обличии. Благодаря элементу ActiveX с точки зрения программиста несколько ускоряется обращение к функциям, тем более что привычнее работать в среде Delphi с компонентами визуально. Вместе с тем визуальность практически не столь важна, т. к. применять приходится те же функции и интерфейсы, а компонент сам по себе не является визуальным, т. е. виден только в среде программирования. Единственное, в чем ускорение заметно, так это в создании объекта. Больше не придется запускать Agent Control через библиотеку COM — элемент управления создается при запуске формы, на которой расположен.

Откройте в Delphi пустое приложение и разместите на форме элемент управления ActiveX. Вид формы во время разработки показан на рис. 2.18.

Теперь создадим обработчик события создания главной формы приложения. Компонент ActiveX в этом случае заметно облегчает работу. Нет необходимости загружать сервер MS Agent, подключаться к серверу, запрашивать до-

полнительные интерфейсы — элемент управления имеет достаточное количество открытых методов для работы со всеми функциями MS Agent.

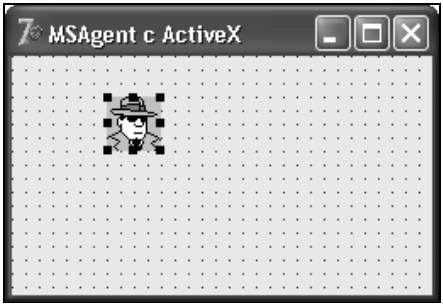


Рис. 2.18. Форма с элементом управления MS Agent во время разработки

Обратимся к коду процедуры создания формы (см. листинг 2.6).

Листинг 2.6. Процедура загрузки персонажа при создании формы с компонентом ActiveX (пример каталога 06_AgentX)

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Agent1.Characters.Load('MyAgent', 'merlin.acs');
  Agent1.Characters.Character('MyAgent').Show(0);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Agent1.Characters.Unload('MyAgent');
end;
```

Посмотрите, как все просто. Элементарным способом загружаем персонаж в коллекцию под именем `MyAgent`, затем сразу отображаем его на экране. В действительности, обращаясь к списку персонажей `Characters`, мы обращаемся к интерфейсу коллекции `IAgentCtlCharacters`, и по вызову его метода `Character` в программу возвращается указатель на интерфейс персонажа `IAgentCtlCharacter` или `IAgentCtlCharacterEx`, в зависимости от того, какой из них был запрошен. Благодаря этому механизму, реализованному в классе `TAgent`, у нас есть возможность вызвать свойства и методы MS Agent в привычном, объектном виде и не думать о получении указателей на интерфейсы через метод `QueryInterface`, хотя и это возможно. Вы, наверное,

заметили, что с Agent Control используются другие интерфейсы, имеющие созвучные названия. Это интерфейсы компонента ActiveX, которые описаны в модуле AgentObjects_TLB, импортированном при установке данного компонента управления, однако все названия аналогичны оригинальным из Agent Server и имеют суффикс `CTL`.

Учитывая то, что код программ с использованием ActiveX несколько меньше и проще, будем рассматривать дальнейшую работу с MS Agent исходя из данного подхода. Ведь ясно, что, разобравшись в интерфейсах, свойствах и методах элемента управления, можно без затруднений перейти на бескомпонентное программирование даже методом позднего связывания.

2.5. Свойства объекта *Request*

Как уже отмечалось в описании объектной модели Microsoft Agent, если для асинхронного запроса к персонажу программист создал переменную для хранения ссылки на объект `Request`, то через нее можно оценить, в каком состоянии находится запрос.

Объект `Request` возвращает следующие методы персонажа: `GestureAt`, `Get`, `Hide`, `Interrupt`, `Load`, `MoveTo`, `Play`, `Show`, `Speak` и `Wait`. Все эти методы мы подробно рассмотрим в *разд. 2.7*.

Для обращения с объектом `Request` служит интерфейс `IAgentCtlRequest`, указатель на который и возвращается указанными выше методами. Какие именно методы, обращенные к персонажу, возвращают указатель на объект `Request`, нет необходимости запоминать. Для их определения всего лишь требуется посмотреть на возвращаемое значение, оно должно иметь тип `IAgentCtlRequest`.

Получение ссылки на объект `Request` увеличивает счетчик ссылок на единицу. В итоге, как и любой COM-объект, он будет удален, когда отпадет необходимость в последней ссылке на него. Отсюда следует, что, если объявить локальную переменную, указывающую на объект, в процедуре или функции, то при выходе за область видимости объект будет автоматически удален, и впоследствии вы не сможете узнать значения его свойств из другого места программы. Поэтому в большинстве случаев необходимо объявлять переменную как глобальную, конечно, если вы не планируете использовать `Request` локально только в одной какой-либо процедуре. При глобальном объявлении объект не будет удален до тех пор, пока программа не завершит работу или переменной не будет присвоено другое значение, т. е. пока она не будет связана с другим объектом.

Интерфейс `IAgentCtlRequest` имеет следующие свойства:

- `ID: Integer`
- `Status: Integer`

- `Number: Integer`
- `Description: WideString`

Свойство `ID` целочисленного типа и по своей сути является счетчиком запросов. У каждого последующего запроса значение свойства на единицу больше, чем у предыдущего. Значение счетчика требуется, когда нужно определить порядок следования запросов, если под каждый запрос отведена собственная переменная. При хранении запросов в одной переменной по идентификатору можно определять, запускались ли уже какие-либо действия персонажа. Если значение не изменилось, значит, запросы к персонажу не поступали, иначе какие-то методы у него были бы запрошены.

Самым важным свойством для оценки состояния запроса является свойство `Status`. Приведу таблицу возможных значений данного свойства (табл. 2.2).

Таблица 2.1. *Описание значений свойства* `Status`

Значение свойства `Status`	**Описание**
0	Запрос успешно завершен
1	Запрос завершился неудачей
2	Запрос находится в очереди
3	Запрос прерван
4	Запрос в процессе выполнения

Объект `Request` включает в себя еще одно целочисленное свойство `Number`, которое хранит код ошибки при выполнении действия или код причины текущего состояния. В то же время свойство `Description` содержит строковое описание, соответствующее коду ошибки или причины состояния. Например, при попытке запуска несуществующей анимации код ошибки помещается в свойство `Number`, а в `Description` — ее описание. В случае прерывания запроса приложением, соответствующим методом, ошибки не возникает, однако свойства `Number` и `Description` будут содержать данные, называемые кодом причины состояния. Разницы между кодом ошибки и кодом причины никакой нет, однако некоторые исключения, не являющиеся исключениями по своему роду создания, иногда называют причиной состояния (имеется в виду состояние свойства `Status`).

Чтобы увидеть своими глазами работу объекта `Request` и оценить ее, рассмотрим демонстрационный пример каталога 07_Request. Вид главного окна демонстрационной программы показан на рис. 2.19.

В программе приведены некоторые примеры (пути) для достижения каждого из возможных состояний объекта `Request`. Для наблюдения за каждым возможным значением проделайте указанные действия. В нижней части глав-

ного окна программы располагается кнопка **GetRequest**, которая отображает текущие значения свойств объекта запросов. Рассмотрим код проекта (листинг 2.7).

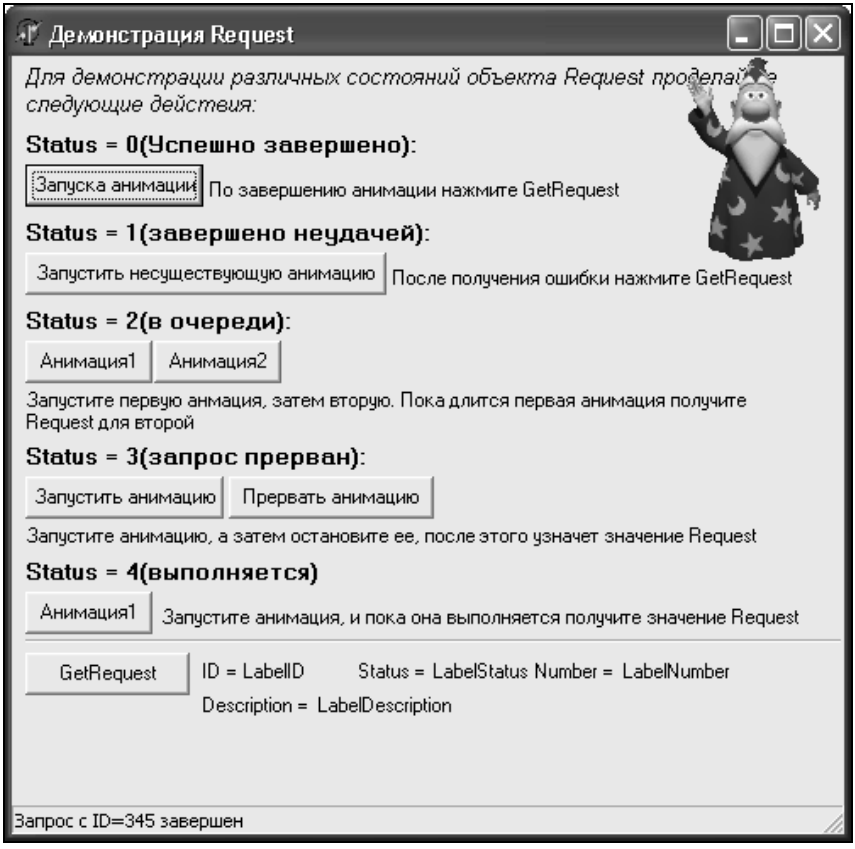


Рис. 2.19. Главное окно программы, демонстрирующей свойства объекта `Request`

Листинг 2.7. Код проекта 07_Request, демонстрирующий применение объекта `Request`

```
unit Unit1;
interface

...
```

```
var
  Form1: TForm1;
  Chars: IAgentCtlCharacterEx;
  Req: IAgentCtlRequest;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  Agent1.RaiseRequestErrors:=false;
  Agent1.Characters.Load('MyAgent', 'merlin.acs');
  Chars:=Agent1.Characters.Character('MyAgent') as IAgentCtlCharacterEx;
  Req:=Chars.Show(0);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Agent1.Characters.Unload('MyAgent');
end;

{**************************************************}
{    Отображение сведений о состоянии запроса      }
{**************************************************}
procedure TForm1.GetRequestClick(Sender: TObject);
begin
 LabelID.Caption:=IntToStr(Req.ID);
 LabelStatus.Caption:=IntToStr(Req.Status);
 LabelNumber.Caption:=IntToStr(Req.Number);
 LabelDescription.Caption:=Req.Description;
end;

{   Демонстрация Status = 0(успешно завершено)     }
procedure TForm1.Button1Click(Sender: TObject);
begin
 Req:=Chars.Play('Wave');
end;
```

```
{   Демонстрация Status = 1(завершено неудачей)    }
procedure TForm1.Button2Click(Sender: TObject);
begin
 Req:=Chars.Play('eklmn');
end;

{   Демонстрация Status = 2(находится в очереди)   }
procedure TForm1.Button3Click(Sender: TObject);
begin
 Chars.Play('DoMagic1');
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
 Req:=Chars.Play('DoMagic2');
end;

{        Демонстрация Status = 3(прервано)         }
procedure TForm1.Button5Click(Sender: TObject);
begin
 Req:=Chars.Play('Search');
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
 Chars.Stop(Req);
end;

{  Демонстрация Status = 4(в процессе выполнения)  }
procedure TForm1.Button7Click(Sender: TObject);
begin
 Req:=Chars.Play('Process');
end;

{          Начало выполнения запроса               }
procedure TForm1.Agent1RequestStart(Sender: TObject;
  const Request: IDispatch);
var
 rq: IAgentCtlRequest;
```

```
begin
 Request.QueryInterface(IID_IAgentCtlRequest, rq);
 StatusBar1.Panels[0].Text:='Запрос с ID='+IntToStr(rq.ID)+' начал обра-
батываться';
end;

{           Конец выполнения запроса                }
procedure TForm1.Agent1RequestComplete(Sender: TObject;
  const Request: IDispatch);
var
 rq: IAgentCtlRequest;
begin
 Request.QueryInterface(IID_IAgentCtlRequest, rq);
 StatusBar1.Panels[0].Text:='Запрос с ID='+IntToStr(rq.ID)+' завершен';
end;

end.
```

Первое, на что следует обратить внимание, это то, что указатель на интерфейс используемого персонажа сохраняется в переменной `Chars` (чтобы каждый раз не обращаться к персонажу через такое длинное обращение), при этом указываем имя интерфейса:

```
Chars:=Agent1.Characters.Character('MyAgent') as IAgentCtlCharacterEx;
```

В коде использованы некоторые незнакомые функции, и я кратко расскажу о них. Сразу бросается в глаза первая строка в процедуре обработки события создания формы приложения TForm1.FormCreate:

```
Agent1.RaiseRequestErrors:=false;
```

Функциональное назначение этого свойства `RaiseRequestErrors` для объекта `Agent Control` будет подробно рассмотрено в следующем разделе, а пока замечу, что в рассмотренной строке кода мы отключаем генерацию исключений, чтобы затем продемонстрировать неудачное выполнение запроса (листинг 2.7, процедура `TForm1.Button2Click`). Если этой строки не будет, то во втором примере (листинг 2.7, процедура `TForm1.Button3Click`), при запуске несуществующей анимации, в обработчике события возникнет исключительная ситуация, которая прервет выполнение кода, и, следовательно, в переменную `Req` не возвратится ссылка на объект `Request`. То есть после нажатия кнопки **GetRequest** мы будем наблюдать состояния предыдущего запроса, об этом можно будет судить по неизменившемуся значению идентификатора `ID`.

Все примеры в данном проекте основаны на проигрывании анимации ассистентом, для чего применяется метод `Play` персонажа, в который передается в виде строкового параметра имя требуемой анимации.

При демонстрации прерывания выполнения запроса (листинг 2.7, процедура `TForm1.Button6Click`) также используется еще незнакомый вам метод `Stop`. Его назначение очевидно — остановка действия для определенного запроса, ссылка которого указывается в его единственном параметре.

Запустите выполнение примера из каталога 07_Request, щелкнув по кнопке `Button1` (**Запуск анимации**), и старичок Merlin примется махать вам рукой. После завершения анимации посмотрите состояние объекта `Request`. Значение свойства `Status` будет равно нулю, что свидетельствует об удачном завершении анимации. Если вы не дождетесь завершения анимации и нажмете кнопку **GetRequest**, то свойство `Status` примет значение 4, т. е. запрос находится в процессе выполнения. Этот же эффект используется для демонстрации последнего примера (листинг 2.7, процедура `TForm1.Button7Click`), однако я решил не совмещать его с первым примером, для разграничения демонстрации различных значений свойства `Status` объекта `Request`.

Продемонстрировать нахождение запроса в очереди возможно, запустив сначала одну анимацию, затем вторую, и пока выполняется первая, получить параметры объекта `Request`. По такому принципу построен третий пример (листинг 2.7, процедура `TForm1.Button3Click`). При запуске первой анимации ссылка на объект `Request` не сохраняется в переменной `Req`, т. к. при щелчке на следующей кнопке ссылка на следующий запрос заменит ее.

Выполняя третий пример (листинг 2.7, процедура `TForm1.Button3Click`) с остановкой запроса, вы увидите, что кроме свойства `Status` изменят свои значения свойства `Number` и `Description`. Мы получим код причины состояния, свидетельствующий о том, что запрос был прерван ничем иным, как нашим собственным приложением.

Упомяну еще об одном важном факте. Вы можете заметить, как при старте выполнения какого-либо запроса или по завершении его выполнения в строке состояния отображается соответствующая информация. Происходит это благодаря обработке двух событий элемента управления: `OnRequestStart` и `OnRequestComplete`. Первое происходит при старте обработки запроса из очереди, а второе — по завершении его выполнения сервером. В виде параметра в обоих событиях возвращается объект `Request`; напрямую мы им не можем воспользоваться, т. к. он представляет собой указатель на интерфейс `IDispatch`, однако запросить необходимый интерфейс `IAgentCtlRequest` можно через метод `QueryInterface`.

2.6. Свойства, события, методы и объекты элемента управления Agent Control

В *разд. 2.4* мы достаточно хорошо познакомились с различными вариантами минимальных приложений с MS Agent. Теперь детально рассмотрим различные части объектной модели Microsoft Agent и подробно поговорим о тех немногих аспектах программирования, непосредственно относящихся к элементу управления Agent Control. В последующих разделах будут описаны другие объекты элемента управления, входящие в него, такие как: `Character Object`, `Balloon Object`, `SpeechInput Object`, `AudioOutput Object` и пр.

Методы

"Методы Agent Control" — это, конечно, громко сказано, т. к. элемент управления обладает лишь одним методом `ShowDefaultCharacterProperties`. Он отображает в указанном месте диалоговое окно для выбора пользователем персонажа по умолчанию, т. е. показывает своеобразную галерею персонажей. Персонажем по умолчанию называется персонаж, выбранный в диалоговом окне **Default Character Properties** (Свойства персонажа по умолчанию). Для загрузки выбранного персонажа вовсе не требуется проводить его поиск в папке с персонажами, его нужно загружать методом `Load` без какого-либо указанного значения в параметре имени файла. В результате будет загружен агент, выбранный в окне **Default Character Properties**. Эта возможность примечательна тем, что не нужно заботиться о сохранении текущего выбранного персонажа и создавать вручную интерфейс пользователя для его выбора. Только нужно иметь в виду, что данная настройка является глобальной и касается всех приложений. Мне кажется, удобнее, если приложение самостоятельно запоминает используемый персонаж, т. к. глобальная настройка может быть изменена в любое время, и ваша программа не сможет проконтролировать изменение персонажа по умолчанию, если она не была загружена, что не гарантирует полную адекватность действий, заложенных в приложении. В конце концов, если пользователю не нравится определенный агент, он всегда сможет его заменить организованными в программе функциями, которые реализуются простейшим способом.

Класс `TAgent` предусматривает три различных варианта вызова метода `ShowDefaultCharacterProperties`: с указанием координат (`X`, `Y`), с указанием только горизонтального положения (координата `X`) и без указания координат (в этом случае окно появляется в прежней позиции). При изменении персонажа пользователем генерируется специальное событие `DefaultCharacterChange` —

для клиентов, загрузивших персонаж по умолчанию. Как уже упоминалось, персонаж можно загрузить методом `Load` без указания какого-либо значения параметра в имени файла. Такие "выкрутасы" свойственны языку Visual Basic, при программировании на котором можно не указывать некоторые параметры, положившись на интерпретатор, способный самостоятельно "додумать" то, что требуется сделать. Например, для Visual Basic возможен такой вариант вызова метода `Load`:

```
Agent.Character.Load "character"
```

И это несмотря на то, что в описании метода `Load` явно указано, что он имеет два параметра, а не один. Delphi не даст нам поступить так, но, тем не менее, подобный вариант возможен в позднем связывании, при котором компилятор не сможет проконтролировать указание всех параметров. Для подобных случаев существует системная переменная `EmptyParam`, которая поможет нам опустить параметр в обычных условиях, т. к. значения `null` или 0 в таких ситуациях не работают.

Приведу соответствующий пример (листинг 2.8, проект каталога 08_DefChar).

Листинг 2.8. Демонстрационный пример `ShowDefaultCharacterProperties`

```
implementation

procedure TForm1.FormCreate(Sender: TObject);
begin
  Agent1.Characters.Load('MyAgent', EmptyParam);
  Agent1.Characters.Character('MyAgent').Show(0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Agent1.ShowDefaultCharacterProperties
end;

procedure TForm1.Agent1DefaultCharacterChange(Sender: TObject;
  const GUID: WideString);
begin
  ShowMessage('Персонаж с GUID='+GUID+' принят персонажем по умолчанию');
end;
```

Замечание

Возможно, вы столкнетесь с исключением при использовании метода `ShowDefaultCharacterProperties`. Так, например, на домашней машине автора в системе Windows 98 "в компании" с Delphi 5 вызов метода `ShowDefaultCharacterProperties` поднимает исключение, диалоговое окно которого изображено на рис. 2.20.

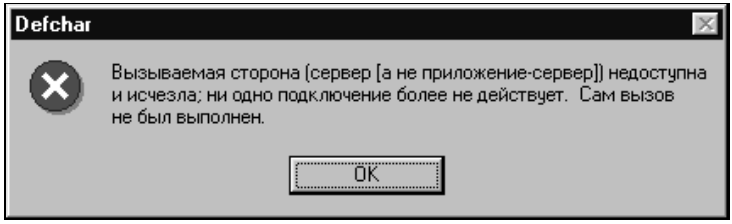


Рис. 2.20. Диалоговое окно с сообщением об ошибке при вызове метода `ShowDefaultCharacterProperties`

Такой же эффект наблюдается и при вызове соответствующего метода из среды Visual Basic 6.0, однако в системах Windows 2000 и Windows XP с Delphi версий 6 и 7 исключение не наблюдается.

Не могу предположить, с чем это связано. Возможно, это происходит вследствие различий в конфигурациях системы, версий библиотек, да и самих сборок Microsoft Agent. Стандартная отговорка, не правда ли?

По этому поводу не стоит расстраиваться, т. к. за всю практику работы с Microsoft Agent я ни разу не оказывался в обстоятельствах, когда невозможно было обойтись без этого метода или события.

Свойства

Непосредственно элемент Agent Control имеет довольно скудный набор свойств. Это продиктовано тем, что все необходимые свойства сосредоточены на персонаже — объекте `Character`. Элемент управления Agent Control (имеется в виду невизуальный компонент) обладает тремя свойствами. Они приводятся в табл. 2.2.

Таблица 2.2. *Свойства элемента управления Agent Control*

Название свойства	Описание
`property Connected: WordBool`	Возвращает или изменяет подключение текущего элемента управления к Agent Server
`property Name: String`	Возвращает или изменяет имя элемента управления

Таблица 2.2 *(окончание)*

Название свойства	Описание
`property RaiseRequestErrors: WordBool`	Показывает, будут ли подниматься исключения при неудачном выполнении запросов. При значении `True` исключения генерируются в случае сбоя. При значении `False` исключения подавляются. По умолчанию свойство имеет значение `True`

С первыми двумя свойствами все более-менее понятно. Остановимся подробнее на последнем — `RaiseRequestErrors`. Из описания видно, что выставляя значение `False`, мы "закрываем глаза" (а вернее, сервер закрывает глаза) на ошибки, вызываемые неудачным выполнением какого-либо запроса к персонажу, т. е. в этом случае сообщения об ошибках не отображаются.

Замечание

Еще раз напоминаю, что определить, какой метод возвращает объект `Request`, можно по возвращаемому результату. Он должен иметь тип `IAgentCtlRequest`.

Установление значения `False` необходимо для языков программирования, не обеспечивающих возможность восстановления после сбоев, к коим Delphi не относится. Тем не менее, это свойство полезно отключать (устанавливать для него значение `False`), если мы предполагаем обрабатывать исключения по-своему (т. е. не через конструкцию `try...except...end`). Другой причиной для отключения свойства `RaiseRequestErrors` может служить попросту нежелание расстраивать конечного пользователя "окнами с красными крестами", возникающими по поводу неудачного завершения действий, выполнение которых несущественно. Мало ли какой персонаж будет применять пользователь? Что выбранный им ассистент поддерживает, а что — нет? Если в программе активно используются запросы к персонажу, выполнение которых не является обязательным, тогда проще всего прописать одну строчку, запрещающую генерацию ошибок при обработке запросов, после чего проблема поднятия исключений будет снята.

Замечание

Не забывайте: при отключении свойства `RaiseRequestErrors` игнорируются только ошибки, связанные с выполнением запросов к персонажу, т. е. только ошибки методов `GestureAt`, `Get`, `Hide`, `Interrupt`, `Load`, `MoveTo`, `Play`, `Show`, `Speak` и `Wait`.

При отладке программы отключение исключений может плохо сказаться на чистоте кода. Если в программе сильно развита система обработки запросов,

как например в случае применения нескольких персонажей одновременно, то довольно трудно будет определить все проникшие в код ошибки.

Продемонстрирую пример кода, показывающий функционирование свойства `RaiseRequestErrors` (см. листинг 2.9) проекта каталога 09_RaiseErrors.

Листинг 2.9. Демонстрация функционирования свойства `RaiseRequestErrors` (пример каталога 09_RaiseErrors). Код секции `implementation`.

```
procedure TForm1.FormCreate (Sender: TObject);
begin
  Agent1.RaiseRequestErrors:=false;
  Agent1.Characters.Load ('MyAgent', 'merlin.acs');
  Agent1.Characters.Character ('MyAgent').Show (0);
end;

{ Генерация ошибки при отсутствии файла с персонажем }
procedure TForm1.Button1Click (Sender: TObject);
var
 Req: IAgentCtlRequest;
begin
 Req:=Agent1.Characters.Load ('MyAgent1', 'eklmn.acs');
 if Req.Status = 1 then
  MessageDlg ('Извините пожалуйста, системе не удается найти файл
персонажа eklmn.acs!',
      mtInformation, [mbOk], 0);
end;

{ Генерация ошибки при отсутствии анимации персонажа }
procedure TForm1.Button2Click (Sender: TObject);
var
 Req: IAgentCtlRequest;
begin
  Req:=Agent1.Characters.Character ('MyAgent').Play ('eklmn');
  if Req.Status = 1 then
   MessageDlg ('Извините пожалуйста, затребованная анимация "eklmn"
отсутствует!',
      mtInformation, [mbOk], 0);
end;
```

```
{ Включатель/Выключатель свойства RaiseRequestErrors }
procedure TForm1.SpeedButton1Click (Sender: TObject);
begin
 Agent1.RaiseRequestErrors:=not Agent1.RaiseRequestErrors;
 SpeedButton1.Down:=SpeedButton1.Down and true;
 if Agent1.RaiseRequestErrors then
  SpeedButton1.Caption:='Отключить генерацию исключений!'
 else
  SpeedButton1.Caption:='Включить генерацию исключений!'
end;
```

На рис. 2.21 показано главное окно приложения. Две верхние кнопки — **Создать исключение** и **Создать исключение 2** — создают два исключения. В коде процедуры обработки события щелчка мышью на первой кнопке (`TForm1.Button1Click`) описана загрузка второго персонажа по несуществующему имени файла. Конечно, если персонаж с именем файла eklmn.acs присутствует в каталоге ассистентов, то исключение в любом случае не возникнет, однако не будем рассматривать данное обстоятельство. По щелчку на второй кнопке (`TForm1.Button2Click`), создающей исключительную ситуацию, производится попытка воспроизведения несуществующей анимации eklmn методом `Play` персонажа с идентификатором `MyAgent`.

Рис. 2.21. Главное окно приложения каталога 08_RaiseErrors

В нижней части главного окна приложения расположена кнопка, включающая и выключающая генерацию исключений, т. е. устанавливающая значение свойства `RaiseRequestErrors` в положение `True` и `False` соответственно. Посмотрите, как работает ошибочный код в обоих вариантах. В случае отключенных исключений (значение `False`) окна с ошибками не появляются как при запуске "из-под" Windows, так и при запуске из среды Delphi, зато появляются "вежливые" сообщения о неполадках. Они организованы с

помощью объекта `Request`, и если запрос закончился неудачей, то соответствующее сообщение выдается. Однако этого не происходит, если значение свойства `RaiseRequestErrors` — `True`. Так как "благодаря" поднятию исключительных ситуаций программа аварийно выходит из процедуры, не завершая ее, поэтому появляются обычные окна с сообщениями об ошибках.

На основании вышеизложенного можно заключить, что когда в программе предусмотрена смена персонажей, то не все они будут поддерживать все разнообразие возможных анимаций. Если вывод сомнительной анимации не особенно важен, т. е. анимацию можно пропустить, то сначала следует отладить приложение с включенной генерацией ошибок, а на завершающей стадии проекта вставить код, отключающий исключения в запросах к персонажу. Допустим, вы пишете программу, имеющую функцию текстового редактора Блокнот, и при вводе пользователем символов заданный персонаж начинает воспроизводить анимацию записи под диктовку, как это делают персонажи Скрепка или Ученый в Word. Естественно, что у части персонажей, существующих в мире, этой анимация нет, а у некоторых из них если и есть что-нибудь подходящее, то под другим названием. Думаю, пользователю не понравится, если при каждом вводе символа будет появляться сообщение об отсутствующей анимации.

Для обработки некоторых ошибок, встречающихся при неудачном выполнении запросов к персонажам, вовсе не нужно включать на время свойство `RaiseRequestErrors`, обрабатывать исключение конструкцией `try... except..end`, а затем отключать свойство вновь. Для этого достаточно воспользоваться объектом `Request`, так как мы это сделали в проекте RaiseErrors (см. листинг 2.9).

События

Элемент Agent Control имеет довольно большое количество разнообразных событий, почти все из которых так или иначе генерируются вследствие деятельности персонажа, под воздействием пользователя на него, а также в результате влияния других приложений, использующих MS Agent. Поэтому невозможно понять смысл сразу всех возможных событий MS Agent.

Для демонстрации некоторых событий рассмотрим проект 10_AgentEvent. Те из событий, которые оказались не рассмотренными и не продемонстрированными, будут описаны в следующих разделах. Здесь не будет показываться внутренняя организация программного кода, т. к. он не отличается сложностью.

Обратим внимание на рабочее окно приложения (рис. 2.22). В проекте у него достаточно скудный графический интерфейс. Так как главная задача проекта — демонстрация функционирования событий, основную свою активность оно проявляет при возникновении определенных событий: текстовые сообщения появляются в списке.

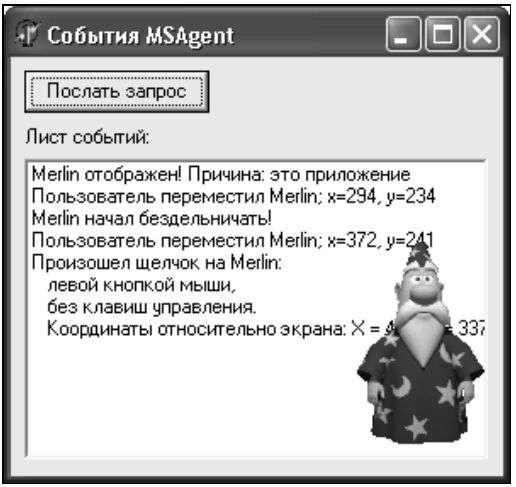


Рис. 2.22. Рабочее окно приложения, демонстрирующего события MS Agent

Полный перечень событий элемента управления MS Agent приведен в табл. 2.3.

Таблица 2.3. *События элемента управления MS Agent*

Событие	Описание
`type TAgentActivateInput = procedure (Sender: TObject; const CharacterID: WideString) of object;` `property OnActivateInput: TAgentActivateInput;`	Происходит, когда персонаж `CharacterID` становится активным для ввода (input-active). Это событие может произойти в тех случаях, когда: персонаж получает щелчок мышью; пользователь называет имя персонажа; при скрытии персонажа другого приложения; при отображении персонажа методом `Show` и т. д. Событие посылается клиенту, для которого персонаж находится в состоянии non-input-active
`type TAgentDeactivateInput = procedure (Sender: TObject; const CharacterID: WideString) of object;` `property OnDeactivateInput: TAgentDeactivateInput;`	Происходит, когда активный персонаж теряет активность ввода. Например, пользователь выбирает мышью другой персонаж, или методом `Show` отображается еще один персонаж. Событие посылается клиенту, для которого персонаж находится в состоянии input-active

Таблица 2.3 *(продолжение)*

Событие	Описание
`type TAgentClick = procedure (Sender: TObject; const CharacterID: WideString; Button: Smallint; Shift: Smallint; x: Smallint; y: Smallint) of object;` `property OnClick: TAgentClick;`	Вызывается при щелчке мышью на персонаже. В виде параметров передаются: идентификатор персонажа, кнопка мыши (`button`), значение управляющей клавиши (`shift`) и координаты относительно экрана (не персонажа). Значения `Button`: 1 — левая кнопка мыши, 2 — правая, 4 — средняя, 4097 — щелчок левой кнопкой на пиктограмме персонажа в системном лотке, 4098 — щелчок правой кнопкой на пиктограмме в TaskBar. Значения `Shift`: 1 — <Shift>, 2 — <Ctrl>, 3 — <Shift>+<Ctrl>, 4 — <Alt>, 5 — <Shif>t+<Alt>, 6 — <Ctrl>+<Alt>, 7 — <Shift>+<Ctrl>+<Al>t, 0 — без клавиши. Событие посылается клиенту, персонаж которого находится в состоянии input-active
`type TAgentDblClick = procedure (Sender: TObject; const CharacterID: WideString; Button: Smallint; Shift: Smallint; x: Smallint; y: Smallint) of object;` `property OnDblClick: TAgentDblClick;`	Вызывается при двойном щелчке на персонаже. Параметры аналогичны параметрам события `OnClick` Событие посылается только активному клиенту, т. е. персонаж которого находится в состоянии input-active
`type TAgentDragStart = procedure (Sender: TObject; const CharacterID: WideString; Button: Smallint; Shift: Smallint; x: Smallint; y: Smallint) of object;` `property OnDragStart: TAgentDragStart;`	Вызывается, когда пользователь начинает перетаскивать персонаж мышью. Событие посылается клиенту, персонаж которого находится в состоянии input-active
`type TAgentDragComplete = procedure (Sender: TObject; const CharacterID: WideString; Button: Smallint; Shift: Smallint; x: Smallint; y: Smallint) of object;` `property OnDragComplete: TAgentDragComplete;`	Запускается, когда пользователь отпускает кнопку мыши после перемещения персонажа. Событие посылается клиенту, персонаж которого находится в состоянии input-active

Таблица 2.3 *(продолжение)*

Событие	Описание
`type TAgentShow = procedure (Sender: TObject; const CharacterID: WideString; Cause: Smallint) of object;` `property OnShow: TAgentShow`	Вызывается при появлении персонажа. Значения `Cause`: 2 — пользователь отобразил персонаж с помощью команды меню или голосом, 4 — ваше приложение отобразило персонаж, 6 — другое клиентское приложение показало персонаж Событие посылается сервером всем клиентским приложениям, использующим персонаж с данным `CharacterID`
`type TAgentHide = procedure (Sender: TObject; const CharacterID: WideString; Cause: Smallint) of object;` `property OnHide: TAgentHide`	Вызывается при скрытии персонажа. Значения `Cause`: 1 — пользователь скрыл персонаж по команде контекстного меню значка персонажа в TaskBar или голосовым вводом, 3 — ваше приложение скрыло персонаж, 5 — другое клиентское приложение скрыло персонаж, 7 — пользователь скрыл персонаж командой контекстного меню пиктограммы персонажа
`type TAgentRequestStart = procedure (Sender: TObject; const Request: IDispatch) of object;` `property OnRequestStart: TAgentRequestStart;`	Посылается сервером, когда он начинает обрабатывать запрос, поставленный в очередь. В виде параметра передается указатель на объект `Request` запроса, который начал обрабатываться. Посылается клиенту, запустившему запрос
`type TAgentRequestComplete = procedure (Sender: TObject; const Request: IDispatch) of object;` `property OnRequestComplete: TAgentRequestComplete;`	Посылается, когда сервер закончил выполнение запроса `Request`. Посылается клиенту, запускавшему запрос
`type TAgentBookmark = procedure (Sender: TObject; BookmarkID: Integer) of object;` `property OnBookmark: TAgentBookmark;`	Вызывается, когда персонаж встречает в читаемом тексте закладку. Закладки создаются тегом `Mrk` *(см. разд. 2.8)*. `BookmarkID` в этом случае содержит целочисленный идентификатор закладки. Вызывается для клиента, читающего текст

Таблица 2.3 *(продолжение)*

Событие	Описание
`type TAgentCommand = procedure (Sender: TObject; const UserInput: IDispatch) of object;` `property OnCommand: TAgentCommand;`	Вызывается, если ваше приложение активно и пользователь выбрал команду персонажа с помощью меню или используя речевой ввод
`type TAgentIdleStart = procedure (Sender: TObject; const CharacterID: WideString) of object;` `property OnIdleStart: TAgentIdleStart;`	Вызывается, когда персонаж с идентификатором `CharacterID` начинает "бездельничать", т. е. перестает выполнять запросы, по истечении периода ожидания переходит к самостоятельным действиям. Возможность перехода к этому состоянию контролирует свойство `IdleOn` объекта `Character` (подробнее *см. разд. 2.7*). Событие посылается всем клиентским приложениям, использующим данный персонаж
`type TAgentIdleComplete = procedure (Sender: TObject; const CharacterID: WideString) of object;` `property OnIdleComplete: TAgentIdleComplete;`	Вызывается, когда персонаж перестает бездельничать, например, когда какое-либо приложение посылает данному персонажу какой-либо запрос. Однако персонаж не перестает бездельничать, если он становится активным для ввода Данное событие сервер также посылает всем клиентам, использующим данный персонаж
`type TAgentMove = procedure (Sender: TObject; const CharacterID: WideString; x: Smallint; y: Smallint; Cause: Smallint) of object;` `property OnMove: TAgentMove;`	Вызывается, если персонаж был перемещен по экрану. Параметры: идентификатор персонажа, новые координаты персонажа по левому верхнему углу относительно экрана и код причины (`Cause`). Значения `Cause`: 1 — пользователь переместил персонаж, 2 — ваше приложение переместило персонаж, 3 — другое клиентское приложение переместило персонаж, 4 — сервер переместил персонаж в связи со сменой разрешения экрана. Событие посылается клиентам, загрузившим этот персонаж

Таблица 2.3 *(продолжение)*

Событие	Описание
`type TAgentSize = procedure (Sender: TObject; const CharacterID: WideString; Width: Smallint; Height: Smallint) of object;` `property OnSize: TAgentSize;`	Происходит, когда изменяется размер персонажа, в этом случае возвращаются новые размеры персонажа. Событие посылается клиентам, загрузившим этот персонаж
`type TAgentBalloonShow = procedure (Sender: TObject; const CharacterID: WideString) of object;` `property OnBalloonShow: TAgentBalloonShow;`	Вызывается при появлении воздушного шарика (Balloon) с читаемым сообщением. Событие посылается клиенту, персонаж которого использовал Balloon
`type TAgentBalloonHide = procedure (Sender: TObject; const CharacterID: WideString) of object;` `property OnBalloonHide: TAgentBalloonHide;`	Событие обратно предыдущему, т. е. происходит при исчезновении воздушного шарика (Balloon). Событие посылается клиенту, персонаж которого использовал Balloon
`type TAgentHelpComplete = procedure (Sender: TObject; const CharacterID: WideString; const Name: WideString; Cause: Smallint) of object;` `property OnHelpComplete: TAgentHelpComplete;`	Вызывается, когда пользователь выходит из контекстно-зависимой справочной службы, запущенной агентом *(см. разд. 2.7)*. В виде параметров передаются: идентификатор персонажа, идентификатор команды (если пользователь вышел из справки через подачу команды), код причины (`Cause`). Возможные значения `Cause`: 1 — пользователь выбрал команду вашего приложения, 2 — пользователь выбрал **Commands Object** другого клиента, 3 — пользователь выбрал команду **Открыть окно команд голосом** (Open Voice Commands), 4 — пользователь выбрал команду **Закрыть окно команд голосом** (Close Voice Commands), 5 — выбрали команду **Показать** (Show), 6 — выбрали команду **Скрыть** (Hide), 7 — пользователь выбрал персонаж (щелкнул на нем)

Таблица 2.3 *(продолжение)*

Событие	Описание
`type TAgentListenStart = procedure (Sender: TObject; const CharacterID: WideString) of object;` `property OnListenStart: TAgentListenStart;`	Вызывается, когда персонаж начинает "слушать" (распознавать речь). Событие посылается всем клиентам. Обычно происходит при нажатии горячей клавиши или программным путем при вызове метода `Listen` персонажа. Событие посылается всем клиентам, использующим персонаж, однако активному клиенту возвращается идентификатор персонажа, а остальным приложениям возвращается пустая строка
`type TAgentListenComplete = procedure (Sender: TObject; const CharacterID: WideString; Cause: Smallint) of object;` `property OnListenComplete: TAgentListenComplete;`	Происходит при завершении распознавания речи. Значения `Cause`: 1 — распознавание было отключено программно, 2 — распознавание, включенное программным кодом, завершено по истечении тайм-аута, 3 — распознавание, включенное горячей клавишей, завершено по истечении тайм-аута, 4 — пользователь отпустил горячую клавишу, 5 — распознавание завершено, потому что пользователь закончил говорить, 6 — клиент, получающий ввод, стал неактивным, 7 — персонаж по умолчанию был изменен, 8 — пользователь отключил речевой ввод. Событие посылается всем клиентам, использующим персонаж, активному клиенту возвращается идентификатор персонажа, остальным приложениям возвращается пустая строка
`type TAgentDefaultCharacterChange = procedure (Sender: TObject; const GUID: WideString) of object;` `property OnDefaultCharacterChange: TAgentDefaultCharacterChange;`	Вызывается при изменении персонажа по умолчанию. Сервер посылает событие только тому клиенту, который загружал персонаж по умолчанию. В виде параметра возвращается GUID нового персонажа по умолчанию
`property OnAgentPropertyChange: TNotifyEvent`	Вызывается, когда пользователь изменяет свойства, касающиеся всех персонажей в диалоговом окне **Дополнительные параметры персонажей** (Advanced Character Options)

Таблица 2.3 *(окончание)*

Событие	Описание
`type TAgentActiveClientChange = procedure (Sender: TObject; const CharacterID: WideString; Active: WordBool) of object;` `property OnActiveClientChange: TAgentActiveClientChange;`	Происходит, когда сменяется активный клиент текущего персонажа. Для клиента, который стал активным, событие генерируется со значением `True` параметра `Active`, для остальных — со значением `False`. Событие может происходить, когда: запускается новый клиент, использующий данный персонаж; при отключении клиента от этого персонажа; в некоторых иных ситуациях

Вы можете заметить в элементе управления доступные события `Restart` и `Shutdown`, однако данные события не поддерживаются MS Agent v2.0, т. е. не посылаются сервером, и присутствуют лишь для обеспечения обратной совместимости.

Объекты

Управляющий элемент Agent Control включает в себя ряд дочерних объектов, а точнее сказать, предоставляет к ним доступ:

- коллекция `Characters Object`
- объект `Character`
- объект `SpeechInput`
- объект `AudioOutput`
- объект `CommandsWindow`
- объект `PropertySheet`

Коллекцию `Characters Object` можно считать рассмотренным элементом, т. к. этот объект имеет всего три метода: `Character` (возвращающий интерфейс персонажа по его идентификатору), `Load` и `Unload`. Все эти методы были подробно рассмотрены в данном разделе и при описании минимальных приложений с MS Agent в *разд 2.4*.

Объект `Character` — элемент коллекции персонажей — подробно описан в следующем разделе *(см. разд. 2.7)*. Объекты `SpeechInput`, `AudioOutput` и `CommandsWindow` будут рассмотрены в *разд. 2.8*.

Объяснять функционирование объекта `PropertySheet`, позволяющего обращаться к некоторым параметрам диалогового окна **Дополнительные парамет-**

ры персонажей (Advanced Character Options), излишне, в нем и так все очевидно, но все же рассмотрим небольшой пример.

Обратиться к окну дополнительных параметров персонажей можно через обращение к свойству `PropertySheet` элемента управления Agent Control:

```
Control.PropertySheet.свойство
```

Например, код для отображения окна свойств будет таким:

```
Agent1.PropertySheet.Left
```

Объект `PropertySheet` имеет в своем распоряжении всего шесть свойств, по двум из которых можно узнать о позиции окна (свойства `Left` и `Top`), по двум другим — о размерах окна (свойства `Height` и `Width`). Я намеренно пишу "узнать", т. к. эти четыре свойства предназначены только для чтения. Осталась неупомянутой последняя пара свойств — `Page` и `Visible`. Свойство `Visible` управляет отображением на экране окна дополнительных параметров персонажей (см. рис. 2.11). При установке свойства `Visible` в значение `True` окно появляется на экране, при изменении значения свойства на `False` — исчезает. Назначение свойства `Page` очевидно из его названия — оно хранит название последней открытой вкладки, однако значение это строковое и имеет три варианта: `'Speech'` (вкладка **Ввод речи**), `'Output'` (вкладка **Вывод**) или `'Copyright'` (вкладка **Авторские права**). Кроме того, свойство `Page` доступно для изменения, т. е. программным кодом можно изменить активную вкладку, а значит, следующий код полностью работоспособен:

```
if Agent1.PropertySheet.Page = 'Speech' then
  Agent1.PropertySheet.Page:='Output';
```

Этот код сработает, если активной вкладкой при последнем запуске была вкладка **Ввод речи** (Speech), т. е. при прошлом закрытии окна она оставалась последней открытой вкладкой. В том случае, если последней открытой вкладкой была вкладка **Вывод** (Output), а пользователь при последующем запуске диалогового окна **Дополнительные параметры персонажей** изменил ее на вкладку **Ввод речи** и не перезагрузил окно, то после выполнения кода вкладка не изменится на обозначенную в коде. Пример работает даже на незапущенном окне дополнительных параметров персонажей, т. е. вкладка изменится, даже когда окно не открыто, но активной вкладкой является **Ввод речи**. Посмотреть работу примера кода можно в проекте каталога 11_PropertySheet.

Для того чтобы оперативно реагировать на изменения настроек, следует обрабатывать событие `OnAgentPropertyChange`. Как только мы щелкаем на кнопке **Применить** или **OK** в окне настройки дополнительных параметров (см. рис. 2.11) после изменения какого-либо параметра, сервер посылает всем клиентам это событие. В демо-приложении в этот момент появляется окно с соответствующим сообщением.

2.7. Объект *Character* элемента управления Agent Control

В предыдущих разделах мы познакомились с минимальными приложениями, использующими анимированный персонаж, а также с методами, свойствами и событиями элемента Agent Control. Как было замечено, почти все события, генерируемые сервером и доступные для обработки элементом управления, непосредственно касаются конкретного персонажа. Мы видели, что почти в каждом событии присутствует параметр `CharacterID`, который определяет идентификатор персонажа, для которого происходит событие.

Все свойства конкретного персонажа сосредоточены на объекте `Character` (интерфейс `IAgentCtlCharacter`, `IAgentCtlCharacterEx`), который доступен как элемент коллекции персонажей (Characters Object) для каждого загруженного агента.

В данном разделе изучим сам объект `Character`. Научимся "знакомиться" с загруженным персонажем программным способом, т. е. узнавать кто он такой, как его зовут и т. д. Затем наш персонаж займет место справочной службы, а далее мы научимся пользоваться всплывающим меню персонажа, его "воздушным шариком" (Balloon). Кроме того, в данном разделе будут рассмотрены все (или почти все) методы и свойства объекта `Character`. С практической точки зрения будет особенно интересен материал подраздела, посвященного получению списка возможных анимаций и, вместе с тем, представляющий классический пример возможностей MS Agent, демонстрируемых во многих компьютерных журналах и статьях, помещенных в Интернете. Не менее значимым станет обзор основных принципов работы с несколькими персонажами одновременно.

Свойства персонажа как элемента управления

MS Agent представляет собой элемент управления, следовательно, как элемент управления (попросту "контрол") он должен обладать хотя бы некоторыми соответствующими свойствами. В данном подразделе, для полноты описываемой картины, мы затронем основные, более-менее очевидные, свойства и методы персонажа.

Итак, персонаж как элемент управления обладает следующими свойствами:

- `Height`
- `Width`
- `OriginalHeight`
- `OriginalWidth`
- `Left`

- `Top`
- `Visible`

Первые два свойства `Height` и `Width` очевидны — это текущие высота и ширина изображения персонажа. Далее по списку следуют менее очевидные свойства — `OriginalHeight` и `OriginalWidth` интерфейса `IAgentCtlCharacterEx`. Их назначение более чем оправданно: т. к. размер персонажа можно изменять через свойства `Height` и `Width`, доступные для чтения и для записи, исходные (оригинальные) значения высоты и ширины агента должны храниться в соответствующих свойствах. Фактически свойства `OriginalHeight` и `OriginalWidth` хранят значения размера кадра, указанного при создании персонажа в Microsoft Agent Character Editor *(см. гл. 4)*.

Свойства `Left` и `Top` имеют свое обычное назначение, как в обычных элементах управления. Они показывают сдвиг в пикселах относительно экрана верхнего левого угла кадра персонажа от левой и верхней границы соответственно. Изменяя их, вы будете перемещать персонаж по экрану.

Свойство `Visible` показывает визуальное состояние, а поскольку оно предназначено только для чтения, то вам не удастся скрыть персонаж установкой свойства в значение `False`.

Для демонстрации свойств объекта `Character` я подготовил приложение, которое доступно в каталоге 12_Prop_Meth. Вид его рабочего окна показан на рис. 2.23.

Рис. 2.23. Демонстрация некоторых свойств объекта `Character`. На рисунке изображен деформированный персонаж Merlin

"Личные" свойства персонажей

Первым делом рассмотрим те свойства, которые присущи именно анимированному помощнику MS Agent как "виртуальной анимированной личности".

Каждый персонаж имеет свое имя, и мы уже видели, как оно отображается во всплывающей подсказке на пиктограмме агента в системном лотке. Имя уже загруженного в коллекцию персонажа можно узнать по свойству `Name` объекта `Character`, например, с помощью метода `ShowMessage`:

```
ShowMessage(Agent1.Characters.Character('MyAgent').Name);
```

Кроме имени, ассистент может иметь небольшое описание, вроде девиза. Намеренно пишу "может иметь", потому что все зависит от воли разработчика, и если при создании персонажа он не пожелал определить описание агента, то девиз будет представлять собой пустую строку. Действительно, описание — это текстовая строка, и получить ее можно через свойство `Description`:

```
Label1.Caption:= Agent1.Characters.Character('MyAgent').Description;
```

Свойства `Name` и `Description` являются основными свойствами, определяющими персонаж как "виртуальную личность", однако существует еще одно свойство, которое могут использовать создатели персонажей, если они не уложились в одну строку описания. Оно называется `ExtraData` и имеет тип `WideString`, как и свойства `Name` и `Description`.

Ранее, в *разд. 2.4*, было сказано, что каждый персонаж также имеет уникальный идентификатор GUID, который можно найти на вкладке **Версия** окна свойств файла ассистента. Программно узнать глобальный идентификатор можно по одноименному свойству `GUID` интерфейса `IagentCtlCharacterEx`, имеющего тип `WideString`.

Следует отметить еще одно свойство — `Version`, возвращающее строку, соответствующую номеру версии стандартного набора анимаций, для которого был откомпилирован персонаж. В принципе, с появлениями новых версий Microsoft Agent расширяется список стандартного набора анимаций. Однако сам редактор персонажей Microsoft Agent Character Editor добавляет информацию о версии при создании нового ассистента. Информация о версии может понадобиться лишь в редких случаях, в основном это касается поддерживаемости различных анимаций.

Для демонстрации перечисленных свойств подготовлен пример программы из каталога 13_AgentProp. Его рабочее окно представлено на рис. 2.24.

В программе организован наиболее простой способ замены персонажей — через раскрывающееся меню. Автор предпочитает применять в своих приложениях именно этот способ, т. к. в данном случае используется минимальное количество компонентов интерфейса и минимальный програмный код. Рассмотрим код программы (листинг 2.10).

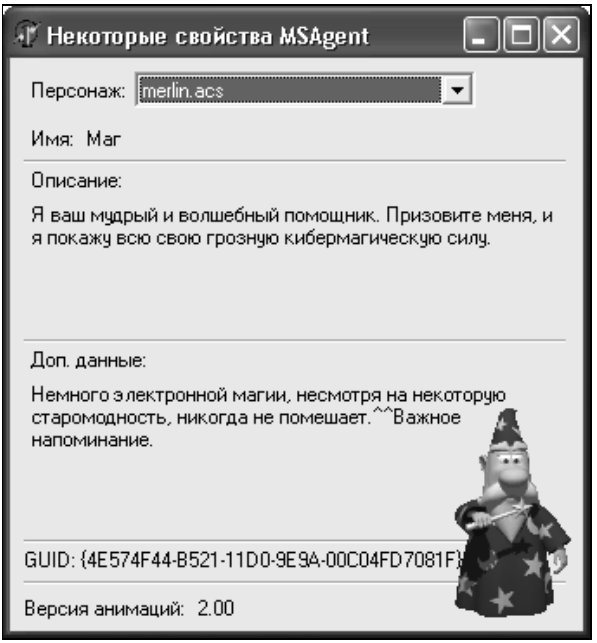


Рис. 2.24. Личные свойства персонажа в окне демонстрационного приложения каталога 13_AgentProp

Листинг 2.10. Код программы примера из каталоге 13_AgentProp (секция `interface` упрощена)

```
unit Unit1;
interface
uses
   ...
type
  TForm1 = class(TForm)
    Agent1: TAgent;
    ...
  end;

var
  Form1: TForm1;
  Chars: IAgentCtlCharacterEx;
```

```
implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
 FiAg: TSearchRec;//Структура будет хранить информацию о найденном файле
 WinDir: array [0..255] of char;//Путь к директории Windows
 AgentFile: string;
begin
  GetWindowsDirectory(WinDir, SizeOf(WinDir));
  //Пытаемся найти первый файл персонажа
  if FindFirst(WinDir+'\MS Agent\chars\*.acs', faAnyFile, FiAg)=0 then
    begin
     cbAgent.Items.Add(FiAg.Name);//Добавляем в список
     cbAgent.ItemIndex:=0;//Устанавливаем выбор на первый персонаж и...
     AgentFile:=FiAg.Name;//...запоминаем имя его файла
     while FindNext(FiAg)=0 do//Пока функция находит следующий файл...
       cbAgent.Items.Add(FiAg.Name);//...добавляем в список
    end
  else
    begin
      MessageDlg('Не найден ни один персонаж MS Agent в '+
         WinDir+'\MS Agent\chars\', mtERror, [mbOk],0);
      halt;
    end;
  cbAgentChange(cbAgent);//Имитируем выбор персонажа из списка
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Agent1.Characters.Unload('MyAgent');
end;

procedure TForm1.cbAgentChange(Sender: TObject);
begin
  if Chars<>nil then//если персонаж был загружен...
   Agent1.Characters.Unload('MyAgent');//...то выгружаем
  Agent1.Characters.Load('MyAgent', cbAgent.Text);
```

```
  Chars:=Agent1.Characters.Character('MyAgent') as IAgentCtlCharacterEx;
  //Отображаем сведения о персонаже:
  lbName.Caption:=Chars.Name;
  lbDes.Caption:=Chars.Description;
  lbData.Caption:=Chars.ExtraData;
  lbGUID.Caption:=Chars.GUID;
  lbVer.Caption:=Chars.Version;
  //Показываем персонаж:
  Chars.Show(0);
end;
end.
```

Обратим внимание на то, как в процедуре обработки события создания формы приложения (`TForm1.FormCreate`) раскрывающийся список заполняется именами файлов доступных персонажей из каталога MS Agent\Chars. В примере используется функция получения расположения каталога системы `GetWindowsDirectory`, а также последовательный поиск файлов с помощью функций `FindFirst` и `FindNext`. Идея алгоритма поиска такова. Ищем первый файл через функцию `FindFirst` по указанному пути с определенными атрибутами (в данном случае с любыми). Результат записывается в специальную структуру типа `TSearchRec`, через которую получаем имя файла (без пути) и добавляем его в список доступных персонажей. Функция `FindFirst` возвращает ноль, если хотя бы один файл был найден. Соответственно, по этому значению определяем, следует ли продолжать поиск. В случае успешного поиска продолжаем его с помощью функции `FindNext`, имеющей всего один выходной параметр — структуру для хранения информации о найденном файле. Все остальные необходимые параметры берутся такими же, как при последнем вызове `FindFirst`.

После поиска доступных персонажей мы вызываем функцию `cbAgentChange`, эмулируя тем самым выбор персонажа из списка (т. к. до этого курсор выбора мы установили в значение 0, т. е. "первая строка").

В силу того, что заменяет персонаж та же процедура обработки события, следует на входе проверять, не равен ли `nil` указатель на персонаж `Chars`. Значение `nil` будет говорить о том, что выгружать персонаж не требуется, т. к. никакой агент попросту не загружен.

Ну, а на последнем этапе работает блок вывода в текстовые метки значений имени, описания и других данных персонажа.

Естественно, что этим списком круг свойств объекта `Character` не исчерпывается, здесь мы рассмотрели только так называемые "личные" его качества. Далее же будем знакомиться с другими свойствами.

Основные свойства и методы объекта *Character*

MS Agent — технология анимированного персонажа, поэтому свою основную работу агент совершает, постоянно демонстрируя какую-либо анимацию. Для того чтобы заставить персонаж выполнять анимацию, не требуется знание большого количества методов. Ранее мы уже сталкивались с методом `Play`, воспроизводящим анимацию по ее строковому имени. Действительно, данный метод чаще других используется в коде, и с ним, пожалуй, может конкурировать еще только пара методов.

Предполагаю, что некоторые читатели, встретившись с методом `Play` при разборе материала предыдущих разделов, могли подумать, что достаточно им научиться определять список анимаций, поддерживаемых персонажем, как они в полном объеме овладеют технологией MS Agent. Не могу согласиться со многими из этих нетерпеливых читателей. Обычно так считают новички или "продвинутые" профессионалы. И если в самом деле нет необходимости долго объяснять некоторые тонкости профессионалам (мне думается, что достаточно сказать им, что MS Agent — это OLE Automation Server), то начинающим программистам будет интересно узнать об этих тонкостях. Естественно, и относительно скромных знаний уже достаточно для того, чтобы писать небольшие приложения, использующие анимированный персонаж как комическую игрушку. При создании более крупных программ, где функции персонажа значительно шире и могут заключаться, например, в интеллектуальной помощи пользователю, у неопытного программиста то и дело начнут возникать вопросы типа: "А вот так можно?", "А как это делается?", "А как обычно поступают таких ситуациях?" и т. д.

Немного отвлекшись, перейдем все-таки к рассмотрению некоторых основных методов объекта `Character`.

Начнем с того, что основное поведение персонажа проявляется через воспроизведение анимации, т. к. он является визуальным элементом управления (имеется в виду сам персонаж, а не компонент Agent Control). Благодаря анимации пользователь узнает о действиях персонажа, о том, на что тот пытается ему указать. Даже речевые возможности по мощности и наглядности нельзя сравнивать с анимацией (обратите внимание, ни в MS Office 2000, ни даже в Office XP не используется речь).

Ассистент начинает воспроизводить анимацию с момента своего появления и не прекращает воспроизведение до момента выгрузки. Мы уже знаем, что получить доступ к анимациям можно через коллекцию `AnimationNames`. Некоторые анимации запускаются самостоятельно сервером MS Agent в случае перехода в одно из определенных состояний, если разработчик назначил анимацию на событие.

Замечание

Имя анимации необязательно совпадает с названием состояния — эта ассоциация устанавливается во время разработки персонажа; более того, несколько различных анимаций могут быть ассоциированы с определенным состоянием для того, чтобы сервер в нужное время выбрал случайным образом одну из них.

Например, при отображении агента методом `Show` и скрытии его методом `Hide` воспроизводится, соответственно, анимация Show или Hide. Кроме того, персонаж Microsoft Agent может находиться в некоторых *состояниях*, таких как: Hearing и Listening (услышал и слушает), Gesturing (указывает), Showing и Hiding (отображение и скрытие), Idling (безделье), Speaking (чтение), Moving (перемещение).

Если разработчик персонажа указал какую-то анимацию, которую следует воспроизводить при переходе персонажа в определенное состояние, то назначенная анимация будет воспроизводиться самостоятельно сервером. В некоторых методах (уже не раз упоминавшихся), таких, например, как `Show` и `Hide`, разработчику дается право выбора, воспроизводить ли заданную анимацию или пропускать ее. Состояние "безделье" также входит в число настраиваемых состояний, правда настраивается оно через специальное свойство `IdleOn`; и понять, почему так происходит, мы сможем, если раскроем смысл данного состояния.

С точки зрения пользователя, ассистент работает, когда выполняет какой-нибудь запрос — например, перемещение, распознавание команд, чтение текста. Это действительно так, и, следовательно, после выполнения заданных действий персонаж может делать что угодно. Как уже отмечалось, при разработке MS Agent корпорация Microsoft большое внимание уделила усилению социальности персонажа, и, как следствие, в свое свободное от запросов время агент бездельничает в прямом смысле этого слова. Это состояние бездействия (Idling) наступает по истечении непродолжительного тайм-аута, наступающего вслед за выполнением каких-либо действий, и в этот момент персонаж выполняет анимацию, ассоциированную с данным состоянием (рис. 2.25). Состояние бездействия бывают трех уровней:

- IdlingLevel1 — достигается сразу после перехода в состояние;
- IdlingLevel2 — через некоторый период времени;
- IdlingLevel3 — в период глубокого бездействия (в этот период стандартные персонажи от Microsoft обычно засыпают и громко храпят).

Конечно, воспроизводимая анимация при бездействии зависит от разработчика персонажа, и если этот персонаж богат анимациями, ассоциированными с данным состоянием, то и поведение его становится более реалистичным.

Ассистент выходит из состояния безделья при поступлении новых запросов, возникновении событий речевого ввода и т. д. Когда помощник начинает

бездельничать, происходит событие `IdleStart`, при выходе — `IdleComplete`. Переходом в это состояние можно управлять, т. е. по желанию разработчик в силах отключить переход к бездействию, и персонаж перестанет засыпать после тайм-аута. Свойство `IdleOn` позволяет это сделать, при значении `True` (по умолчанию) сервер персонажа допускает переход к бездействию, при значении `False` — нет. Однако автор не видит смысла отказываться от состояния безделья ассистента, ведь так забавно видеть, как он самостоятельно осматривает вокруг себя мнимое пространство, потягивается, зевает и т. д. Бесспорно, все возможные анимации, проигрываемые свободным от дел персонажем, зависят от его создателя (чем уж он там свое дитя наградил?); так, например, самые распространенные помощники от Microsoft: Маг (Merlin), Джинн (Genie), Робот (Robby) и Попугай (Peedy) достаточно "продвинуты" в этом отношении.

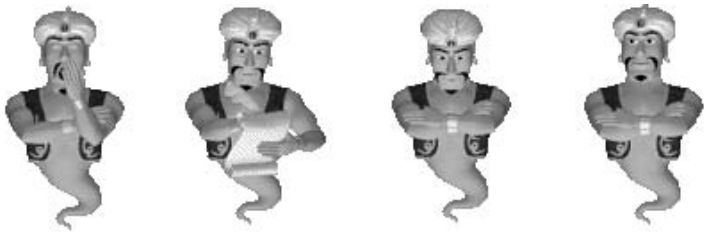


Рис. 2.25. Различные анимации персонажа Genie при бездействии

Не все анимации ассистента должны быть приспособлены для специальных состояний, поэтому может существовать большое количество самых разнообразных и специфичных для конкретного образа анимаций. Все они воспроизводятся специальным методом `Play`, имеющим всего один параметр (а больше и не надо), представляющий собой строковое имя анимации:

```
Agent1.Characters.Character('MyAgent').Play('wave');
```

Как мы ранее уже отмечали[3] данный метод возвращает объект `Request`.

Существует немало случаев применения MS Agent, в которых персонаж выступает в роли интерактивного помощника. В этих и других ситуациях довольно часто можно столкнуться с трудностями в реализации анимации указывания персонажем на какую-либо точку экрана. Слишком накладно проверять, в какой точке он находится, перемещать его, а затем рассчитывать, в какую сторону следует указать рукой и, наконец, воспроизводить

[3] *См. разд. 2.5* данной главы. Немного опережающее описание вполне оправданно, смысл некоторых методов более чем очевиден, а т. к. в некоторых предыдущих примерах нельзя было обойтись без определенных методов, то автор и выбрал такой принцип изложения материала.

нужную анимацию. Для упрощения подобных манипуляций с персонажем служит метод `GestureAt`, который обладает двумя целочисленными параметрами для указания необходимой точки экрана:

```
Agent1.Characters.Character('MyAgent').GestureAt(500, 300);
```

В приведенном примере персонаж укажет руками на соответствующую область экрана с координатами $x = 500$ и $y = 300$, исходя из своего текущего местоположения. Если точка находится слева от персонажа, то и руки он будет обращать в левую сторону; в других комбинациях сервер автоматически определит, какую анимацию нужно воспроизводить, чтобы ассистент показал в нужную сторону (рис. 2.26).

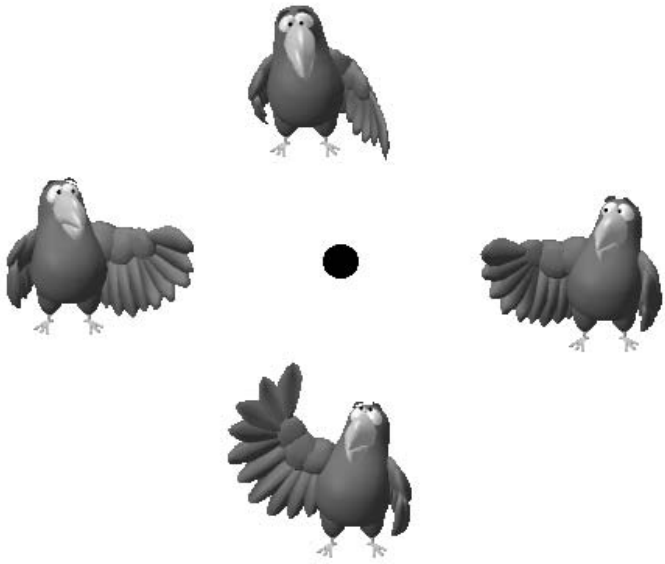


Рис. 2.26. Пример того, как персонаж Peedy указывает из четырех различных положений на одну и ту же точку (черное пятно — мнимая точка). Все анимации запускаются одним методом (`GestureAt`), при этом сервер сам вычисляет, какую из четырех анимаций следует воспроизводить

Таким образом, нам не стоит заботиться о каких-то вычислениях, но все же желательно, чтобы персонаж располагался на небольшом расстоянии от места, на которое следует указывать, т. к. при больших расстояниях пользователю трудно будет понять, на что персонаж пытается обратить его внимание, ведь анимаций указания всего четыре: влево, вправо, вверх, вниз. Для этих целей используется еще один метод, о котором сейчас пойдет речь.

Как мы понимаем, пользователь может перемещать персонаж мышью, программно же изменять его местоположение возможно, непосредственно меняя координаты через свойства `Left` и `Top`. Однако для более эффектного перемещения персонажа программным способом используется специальный метод `MoveTo` объекта `Character` интерфейса `IAgentCtlCharacter`:

```
function  MoveTo(x: Smallint; y: Smallint; Speed: OleVariant):
IAgentCtlRequest;
```

При его выполнении также запускается анимация, ассоциированная с данным состоянием персонажа. Первые два параметра очевидны: это координаты нового местоположения агента (верхнего левого угла его формы[4]) относительно экрана, третий параметр показывает скорость перемещения персонажа по экрану, а именно то, как быстро изменяется шаг передвижения персонажа. Возможные значения этого параметра начинаются с нуля, ноль при этом фактически означает пропуск анимации (т. е. персонаж появляется в новом месте мгновенно). При остальных значениях персонаж перемещается с задержкой и одновременным воспроизведением анимации. Хотя параметр скорости перемещения указывается в миллисекундах, это достаточно условно, т. к. даже при значениях 1 и 1000 разница по скорости в 1000 раз не ощущается. Перемещение совершается исходя из мощности компьютера и степени загруженности его другими задачами.

Метод `MoveTo` достаточно часто выполняется перед отображением персонажа при его загрузке, например:

```
var
  Req: IAgentCtlRequest;
  Chars: IAgentCtlCharacterEx;
begin
  Agent1.Characters.Load('MyAgent', 'Merlin.acs');
  Chars:=Agent1.Characters.Character('MyAgent') as IAgentCtlCharacterEx;
  Req:=Chars.MoveTo(100, 100, 0);
  Chars.Show(0);
end;
```

При этом, если вы сохраняете объект `Request`, то проследите, чтобы метод `MoveTo` не выполнялся со значением параметра скорости, не равным нулю (т. е. в такой ситуации всегда должен быть 0), иначе в свойстве `Status` объекта `Request` вы получите значение 1 (завершено неудачей), в свойстве `Description` — сообщение об ошибке, а именно о том, что была произведена попытка воспроизведения анимации при скрытом персонаже. Сообщение

---

[4] Иначе говоря — не форма агента, а кадр, т. к. агент строится из анимаций, а анимация состоит из отдельных кадров.

с "красным крестом" не появится, т. к. причина в состоянии, однако мы должны о нем помнить, если ошибка может как-то повлиять на дальнейшее выполнение кода.

Применяя метод `MoveTo`, сервер генерирует событие `OnMove` для элемента управления, так же как и при перемещении персонажа пользователем с помощью мыши. Параметр события `Cause` расскажет нам о способе перемещения. Однако пользоваться этим параметром можно только в процедуре обработки событий. Что же делать, если из другого места программы требуется узнать, кто или что переместило персонаж? В этих и других случаях пользуются свойством `MoveCause` объекта `Character` интерфейса `IAgentCtlCharacter`, имеющим тот же спектр значений, что и параметр `Cause` события `OnMove` если персонаж был перемещен, и значение 0 — если персонаж не передвигали (подробное описание значений свойства `MoveCause` приведено в табл. 2.4). Замечу также, что это свойство широко применяется при контроле изменения местоположения персонажа в случае использования его несколькими приложениями, т. к. отдельные решения не всегда удобно реализовывать в обработчиках событий. Например, в каком-то месте программного кода вашего приложения в нужное время требуется воспроизвести определенную анимацию персонажа, при этом другое приложение (а возможно, и пользователь) всегда может изменить положение этого персонажа. Понятно, что для достижения успеха следует контролировать причину перемещения персонажа с помощью свойства `MoveCause`.

Таблица 2.4. *Описания значений свойства* `MoveCause`

Значение `MoveCause`	**Описание**
0	Персонаж не перемещался
1	Персонаж перемещен пользователем
2	Ваше приложение переместило персонаж
3	Другое клиентское приложение переместило персонаж
4	Персонаж перемещен сервером в связи со сменой разрешения экрана

Помимо свойства `MoveCause` (показывающего причину перемещения персонажа), существует аналогичное свойство целочисленного типа, по значению которого можно определить причину текущего состояния видимости — `VisibilityCause`. Множество значений, возможных для данного свойства, получается путем объединения множеств допустимых значений параметра `Cause` для событий `OnShow` и `OnHide` элемента управления и значения 0 (если персонаж не отображался). Все значения представлены в табл. 2.5.

Таблица 2.5. *Описание значений свойства* `VisibilityCause`

Значение `VisibilityCause`	**Описание**
0	Персонаж не отображался
1	Пользователь скрыл персонаж по команде всплывающего меню значка персонажа в TaskBar или голосовым вводом
2	Пользователь отобразил персонаж с помощью команды меню или голосом
3	Ваше приложение скрыло персонаж
4	Ваше приложение отобразило персонаж
5	Другое клиентское приложение скрыло персонаж
6	Другое клиентское приложение показало персонаж
7	Пользователь скрыл персонаж командой контекстного меню персонажа, а не через команду меню на значке в системном лотке

Метод `Think` должен быть знаком читателям по *разд. 2.6* из примера 10_AgentEvent, однако для порядка вспомним и о нем. По его названию можно догадаться, что он заставляет персонаж "задуматься" над чем-нибудь. "Предмет обдумывания" мы указываем в строковом параметре метода — это текст, отображаемый в "воздушном шарике". Если перед вызовом метода запустить анимацию Think, то персонаж будет "думать" весьма правдоподобно (рис. 2.27).

Рис. 2.27. Некоторые последовательные действия с персонажем создают эффект синхронного выполнения

Добиться такого эффекта можно с помощью следующего кода:

```
Agent1.Characters.Character('MyAgent').Play('Think');
(Agent1.Characters.Character('MyAgent') as IagentCtlCharacterEx).Think
('О чем же я сейчас '+ 'думал...');
```

Замечание

Читатели могут удивиться тому, что во второй строке кода явно указывается интерфейс `IAgentCtlCharacterEx`. В Delphi 5 его действительно можно не указывать, а вот в остальных версиях среды придется его определить, т. к. в противном случае при компиляции выдается сообщение о том, что метод `Think` не объявлен.

Возможно, приведенный код покажется вам несколько странным: мы сначала запустили в очередь выполнение анимации, а затем заставили персонаж думать. Получается, что хотя действия должны выполняться последовательно, этого не происходит, и все благодаря тому, что метод `Think`, а также его "брат" — метод `Speak` изменяют состояние персонажа, не затрагивая анимацию. Все происходит правильно, это можно проверить через два объекта `Request`, созданных для двух различных запросов. Если посмотреть значения в самом начале действия, в тот момент, как персонаж только начнет поднимать руку к подбородку, то можно увидеть, что первый запрос на выполнение анимации находится в состоянии 4 (в процессе выполнения), а второй запрос (на выноску) — в значении 2 (в очереди). Параметр состояния `State` обоих запросов определим чуть позже (когда появится воздушный шарик "с мыслями"). Параметр состояния запроса `Play` станет равен нулю (успешно завершен), а метода `Think` — следовательно, 4. Однако видимая синхронность наблюдается, потому что в ситуации тайм-аута между окончанием выполнения запроса и началом "безделья" агента сервер не изменяет анимационного состояния персонажа и продолжает воспроизведение последнего кадра анимации.

Демонстрирует эту особенность, а также другие свойства и методы, которые обсуждались в данном подразделе, проект из каталога 14_AgentMeth (окно приложения приведено на рис. 2.28). В этом приложении мы сначала рассмотрели состояния персонажа (указывания, перемещения, отображения и скрытия и т. д.) и узнали, что, применяя к скрытому персонажу метод `MoveTo` с ненулевой скоростью перемещения, мы будем получать в объекте `Request` ошибку, вернее причину состояния. Попробуем это увидеть. В проекте, представленном на компакт-диске, при загрузке персонажа в процедуре обработки события создания формы приложения как раз и присутствует некорректный запуск `MoveTo`. Мы этого не ощущаем, но если щелкнуть на кнопке **Получить Request** (см. рис. 2.28) сразу после загрузки и отображения персонажа, то мы получим соответствующее сообщение, подтверждающее неадекватность действий.

Рис. 2.28. Рабочее окно приложения, демонстрирующего некоторые методы MS Agent

Следующая кнопка приложения (в порядке "сверху вниз") включает и отключает возможность перехода ассистента в состояние безделья. Посмотрите, как замораживается персонаж на текущем кадре при внезапном отключении свойства `IdleOn`. А тестируя с помощью одноименной кнопки метод `GestureAt`, вы увидите, как влияет положение персонажа относительно указываемой точки на воспроизводимую анимацю. Несмотря на то, что возможных анимаций всего четыре, посмотрите, как эффектно выглядят указания!

Удобно изучать реакцию персонажа на применение методов `MoveTo` и `GestureAt`, заполняя требуемые параметры собственноручно (наример, параметры координат и скорости перемещения в методе `MoveTo`), однако некоторые читатели могут возразить, что демонстрировать функционирование метода `Play` путем ручного ввода названия анимации неинтересно. На этот счет могу заметить, что получение списка возможных анимаций конкретного ассистента — тема *подразд. "Получаем список анимаций" данной главы.*

К свойствам и методам объекта `Character` мы еще неоднократно будем возвращаться, а напоследок замечу, что одно из свойств этого объекта в новой версии MS Agent не поддерживается и существует только для обратной совместимости. Это свойство `Suspended` логического типа, всегда имеющее значение `False`.

Доступ к персонажам через Интернет

Некоторые читатели, посмотрев на название подраздела, подумают, что сейчас пойдет речь об Интернете и web-программировании с помощью Visual

Basic и HTML, а кое-кто может обеспокоиться отсутствием выхода во Всемирную сеть. Предположение это верно наполовину: дело в том, что мы действительно займемся сейчас тем, что научимся получать доступ к персонажам через Интернет, однако делать мы это будем с помощью среды Delphi, так что демонстрационные примеры можно будет проверить без использования Всемирной паутины и модема.

В *разд. 2.1*, посвященном описанию технологии MS Agent, мы говорили о различных форматах файлов персонажей, в том числе о структурном файле формата ACS, которым мы до этого пользовались и в котором обычно распространяются готовые персонажы, и о раздельном формате, разбиваемом на файлы форматов ACF и ACA. Так вот, последний тип файлов широко используется в Интернете (применять его имеет смысл именно во Всемирной сети). Хотя web-страницу с поддержкой технологии MS Agent несложно создать, ее посетитель в большинстве случаев либо вообще не увидит этой экзотики по причине отключенной загрузки ActiveX-компонентов, либо для него окажется слишком накладным загружать все необходимые модули и компоненты. Подумать только, около 400 Кбайт занимает сам MS Agent, а еще персонаж с полным комплектом качественных анимаций — от 1 Мбайт и выше.

Не будем расстраиваться из-за подобных проблем, а обратим внимание вот на что. Хотя Интернет штука дорогая и использовать MS Agent на своих страницах можно пока лишь в качестве необычного мультимедийного средства, которым можно похвастаться перед друзьями (мол, вот что у меня на сайте имеется), все же полезно будет рассмотреть некоторые принципы работы с разделенными форматами персонажей. Естественно, все описания будут даваться на Delphi, и, создавая наши программы, мы попытаемся научиться загружать персонажи (вернее, их данные) прямо из Сети. Можно сгоряча заключить, что приобретенные навыки принесут пользу только на ниве создания web-страниц и нет смысла рассматривать процесс из Delphi, тем не менее, без подобной информации руководство, на роль которого претендует данная книга, будет неполным. К тому же полученные знания легко перенести в web-среду, взяв за основу любое минимальное приложение, использующее MS Agent на VBScript, и нарастив его нужными функциями.

Различие между форматами уже слышится в их названиях. ACS-файл называется структурным файлом, т. к. хранит и описание персонажа, и данные анимаций. ACF-файлы и ACA-файлы представляют собой раздельный формат: так, описание персонажа, его личные свойства хранятся в небольшом файле ACF, а вот данные анимаций — в файле ACA, причем каждая анимация хранится в отдельном файле. Благодаря подобному разделению анимированный персонаж можно использовать в сетевых приложениях, ведь незачем загружать целиком объемный[5] файл ACS, если в течение сеанса работы

[5] Качественные персонажи занимают память объемом более 3 Мбайт.

с MS Agent потребуется всего несколько анимаций, которые все уложатся в несколько десятков килобайт.

Особой разницы в использовании персонажей из файлов формата ACS и файлов раздельного формата нет; более того, если вы планируете использовать раздельный формат локально на одном компьютере, то никаких изменений в коде производить не потребуется. В случае же нахождения разделенного файла где-то на web-сервере, приспособив приложение для работы с этим форматом, нельзя будет в дальнейшем без дополнительных изменений в коде перейти на другой формат. На сегодняшний день еще отсутствуют какие-либо средства от Microsoft для конвертации одного формата в другой и для редактирования собранных агентов, а энтузиасты, которые смогут изучить строение формата файлов и написать конвертер, вряд ли найдутся. Да и зачем это надо, ведь в основном большинство агентов работают в обычных (если хотите, "офисных") приложениях и, следовательно, удобнее пользоваться ACS-форматом.

Собрать персонаж определенного формата можно только в специальной утилите Microsoft Agent Character Editor (о ней мы поговорим в *гл. 4*). Разделенный файл персонажа вполне доступен, если имеются исходный набор изображений ("картинок") и редактор персонажей. Из набора картинок я, например, собрал простой персонаж, нарисованный в редакторе MS Paint и доработанный в Adobe Photoshop. Мы пока не будем рассматривать подробности создания агента с помощью редактора персонажей, это не цель данного раздела.

Итак, что же нам понадобится в ходе рассмотрения примеров? Так как проводить тестирование всех программ через Интернет, т. е. закачивать на сервер необходимые файлы — весьма трудоемкий процесс, то следует прибегнуть к способу, состоящему в создании на локальном компьютере маленького web-сервера. Научившись работать в таких условиях, мы без проблем сможем перейти на любой существующий интернет-сервер. Для этого следует воспользоваться уже существующими web-серверами сторонних разработчиков, такими как сервер Apache HTTP Server — наиболее популярный, распространяемый бесплатно по адресу **http://www.apache.org**, — или маленьким и простым, но обладающим большим количеством функций сервером Small HTTP Server от Максима Феоктистова (**http://srv.mf.inc.ru/**), также бесплатным (однако только для граждан бывшего СССР). Пользователям Windows 2000, а также других серверных операционных систем лучше использовать, пожалуй, службу Internet Information Services (IIS). В отсутствие под рукой какого-либо сервера, а также при нежелании его найти и настроить, можно воспользоваться простейшим web-сервером, который распространяется в виде демо-примера в поставке Delphi 6.

Описание настройки различных серверов выходит за рамки данной книги, документацию по любому из них можно найти в соответствующей литерату-

ре. По серверу Apache существует немало русскоязычных материалов, а для пользователей IIS доступна большая справка. Поэтому в дальнейшем описании я буду ориентироваться на сервер из демо-примеров Delphi 6. Он располагается по адресу $(Delphi6)\Demos\Indy\HTTPServer\, где $(Delphi6) — директория с установленной средой программирования. Если вы не работаете в Delphi 6 или у вас нет этого примера, то воспользуйтесь проектом, расположенным на прилагаемом компакт-диске, в каталоге программ, в папке HTTPServer. Для тестирования примеров я немного упростил интерфейс сервера, скрыл лишние его элементы и добавил кнопку выбора корневой директории документов сервера через стандартное диалоговое окно выбора папок (рис. 2.29).

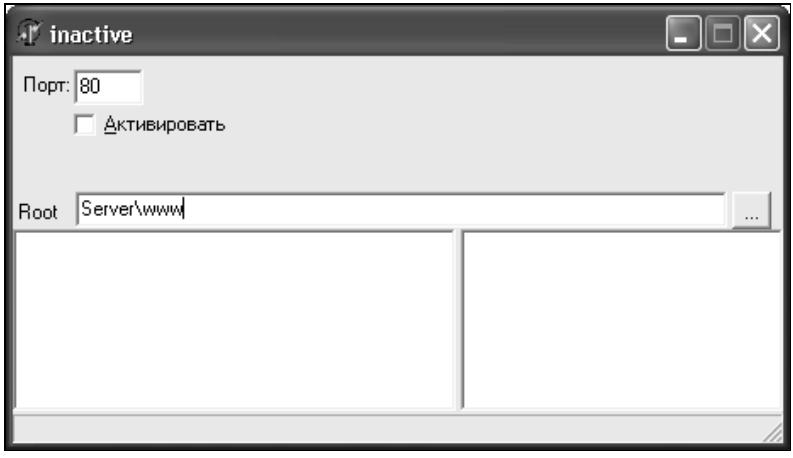


Рис. 2.29. Окно маленького web-сервера, прилагаемого в виде примера к Delphi. На рисунке изображено главное окно приложения, слегка исправленное автором

Для начала проверим работоспособность проекта. Если вы знаете, что используемый вами web-сервер настроен и прекрасно работает, то можно пропустить следующие несколько абзацев.

Основные принципы настроек везде одинаковы, и никакие специальные опции нам не потребуются. Главное, что нам нужно, — это какой-либо виртуальный каталог. Корневой серверный каталог Apache обычно располагается по адресу C:\Program Files\Apache Group\Apache\htdocs, для IIS — это C:\Inetpub\. В нашем простейшем web-сервере корневой каталог, который будет опубликован в web, выбирается кнопкой с троеточием (...), расположенной справа от поля редактирования **Root**. Чтобы долго не думать над содержанием этого каталога, скопируйте папку \www из подкаталога проекта 15_WebAgent на свой жесткий диск, пусть конечное расположение будет:

C:\Server\www. Теперь этот каталог можно выбрать в качестве корневого каталога сервера.

Теперь активируйте web-сервер, установив флажок **Активировать**. Запустите любой имеющийся у вас на компьютере браузер, например, Internet Explorer, Opera или Mozilla, и в строке адреса введите:

```
http://localhost/
```

Должна загрузиться web-страница с приветствием, как показано на рис. 2.30.

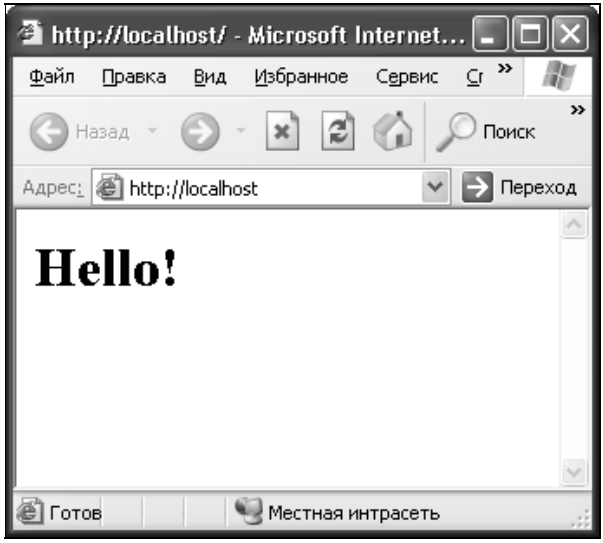


Рис. 2.30. Тестовая web-страница в Internet Explorer, проверяющая работу нашего web-сервера

Если запуск прошел успешно, можно продолжить чтение книги, в противном случае проверьте корректность ваших действий.

Должен сказать, что открытия новых методов загрузки персонажей не будет. Загружаются персонажи из разделенных файлов так же, как всегда: методом `Load`, однако указывать в этом случает нужно не файл ACS, а файл данных о персонаже формата ACF вместе с адресом его расположения. Если вы не копировали папку \www из каталога 15_WebAgent, то скопируйте папку \www\MS Agent в назначенную вами корневую папку сервера. Таким образом, если виртуальный каталог имеет реальный путь C:\Server\www, в него вы должны поместить папку 15_WebAgent\www\MS Agent с прилагаемого компакт-диска, чтобы документы MS Agent\ были доступны по адресу:

http://localhost/MS Agent/

Замечание

Неважно, конечно, по какому адресу видна папка MS Agent, главное, чтобы вы знали URL, по которому доступно ее содержимое. Естественно, в демонстрационном примере вам придется вводить свой собственный адрес для загрузки персонажа.

В папке MS Agent уже лежат все необходимые файлы для примеров — собственно, это единственный персонаж Simple Agent в раздельном формате, т. е. один файл SimpleAgent.acf и несколько ACA-файлов по названию анимаций.

Теперь для загрузки демонстрационного персонажа следует выполнить код:

```
Agent1.Characters.Load('MyAgent',
            'http://localhost/MS Agent/SimpleAgent.acf');
```

Как обычно, первый параметр метода `Load` является идентификатором персонажа, а второй — адресом для загрузки персонажа из каталога web-сервера.

Основываясь на опыте построения предыдущих приложений, вслед за этим уже можно отображать персонаж на экране методом `Show`. Однако если мы сейчас его выполним, то ровным счетом ничего не произойдет. Точнее, пиктограмма персонажа в системном лотке появится (она появляется уже после использования `Load`), а вот сам персонаж на экране не отобразится. Если мы обратимся к состоянию запроса, то обнаружим причину такого положения: "Отсутствует анимация для данного состояния. Убедитесь, что для него назначена анимация". Произошла же такая оказия потому, что нами используется уже другой формат персонажа. Прежде, в ACS-файле, вместе с данными персонажа хранились и его анимации, теперь же мы загрузили только данные из ACF-файла. Поэтому следует дополнительно запросить определенную анимацию для ее воспроизведения. Все действия по загрузке анимации осуществляет метод `Get` объекта `Character` интерфейса `IAgentCtlCharacter`:

```
function  Get(const Type_: WideString; const Name: WideString;
     Queue: OleVariant): IAgentCtlRequest;
```

Данный метод отыскивает специальную анимацию, мультипликации состояний агента или звуковые эффекты для персонажа по их именам и помещает в кэш (в ту папку, где размещаются временные файлы Интернета), затем оттуда они и воспроизводятся. То, каким образом метод, воспроизводящий анимацию, узнает, где располагается файл анимации, нас не должно волновать — раз это скрыто от разработчика, значит, незачем забивать себе голову лишними вопросами, главное, что загруженную анимацию сервер MS Agent всегда находит.

На требуемых значениях параметров метода `Get` следует остановиться подробнее. С первым строковым параметром `Type` нужно указать тип данных, которые требуется загружать. Возможные значения параметра приведены в табл. 2.6.

Таблица 2.6. *Описание значений параметра* `Type` *метода* `Get`

Значение	Описание
`Animation`	Конкретная анимация для персонажа
`State`	Анимации какого-либо состояния персонажа. Так как для одного состояния может быть назначено несколько анимаций, то при указании этого значения для выбранного состояния загружаются все анимации
`WaveFile`	Аудиофайл для персонажа, который можно будет использовать при чтении вместо синтезируемой речи

Очевидно, что от выбранного значения первого параметра будет зависеть круг значений следующего параметра `Name`. Если для загрузки определена конкретная анимация (т. е. `Type_ = Animation`), то второй параметр должен иметь строковое имя нужной мультипликации.

```
Agent1.Characters.Character('MyAgent').Get('Animation', 'Wave', true);
```

Указывая загрузку анимаций состояния агента через параметр `Type`, во втором параметре — `Name` — запрашивают требуемое состояние. Список запрашиваемых состояний приведен в табл. 2.7.

Таблица 2.7. *Список запрашиваемых состояний*

Значение	Описание
`Gesturing`	Все указательные анимации
`GesturingDown`	Анимации для указания персонажем в напралении вниз
`GesturingLeft`	Анимации для указания персонажем налево (со стороны персонажа или экрана, т. е. для нас направо)
`GesturingRight`	Анимации для указания персонажем направо
`GesturingUp`	Анимации для указания персонажем вверх
`Hiding`	Анимации исчезновения персонажа
`Hearing`	Анимации для состояния, когда персонаж слушает команды пользователя
`Idling`	Анимации всех уровней бездействия

Таблица 2.7 *(окончание)*

Значение	Описание
`IdlingLevel1`	Анимации бездействия первого уровня (проигрываемые сразу после начала бездействия)
`IdlingLevel2`	Анимации второго уровня бездействия (проигрываемые по истечении небольшого периода времени)
`IdlingLevel3`	Анимации третьего уровня бездействия (проигрываемые после долгого ожидания активности пользователя)
`Listening`	Анимации ситуаций, когда персонаж только начинает распознавать речевые команды (обычно эти анимации несколько длинней, чем Hearing)
`Moving`	Все анимации перемещения персонажа
`MovingDown`	Анимации перемещения персонажа вниз
`MovingLeft`	Анимации перемещения персонажа влево
`MovingRight`	Анимации перемещения персонажа вправо
`MovingUp`	Анимации перемещения персонажа вверх
`Showing`	Анимации, ассоциированные с появлением персонажа
`Speaking`	Анимации, связанные с воспроизведением речи

Например, следующий код отыскивает все анимации, ассоциированные с состоянием появления персонажа:

```
Agent1.Characters.Character('MyAgent').Get('State', 'Showing', true);
```

Приятная особенность, которой обладает метод `Get`, состоит в том, что запрашивать анимации или состояния не обязательно по одной (одному), список необходимых значений в параметре `Name` можно указывать через запятую:

```
Agent1.Characters.Character('MyAgent').Get('Animation', 'Wave, DoMagic1',
true);
```

или:

```
Agent1.Characters.Character('MyAgent').Get('State', 'Showing, Hiding',
true);
```

Единственное, что недопустимо при вызове метода `Get`, — это смешивать типы требуемых данных. Это означает, что в типе данных нельзя одновременно указать и `Animation`, и `State`. Однако если заявить в списке имен анимаций или состояний несуществующее название, ровным счетом ничего не произойдет: просто загрузятся все существующие анимации или состоя-

ния, а неверные пропустятся (правда, при анализе свойств `Status` и `Description` объекта `Request` будет обнаруживаться сообщение об ошибке). В следующем примере данные существующей анимации Pleasure будут загружены, хотя перед ней стоит название отсутствующей анимации Idling:

```
Character.Get('Animation', 'Idling, Pleasure', true);
```

Волновой (лингвистический или аудио) файл удобно применять в различных ситуациях, например, в отсутствие возможности синтеза речи (при недоступности определенных компонентов) можно загрузить волновой файл, а затем проиграть его, одновременно отображая нужные слова в тексте "воздушного шарика". Этот файл должен находиться в каталоге сервера и загружаться с указанием его адреса:

```
Character.Get('WaveFile', 'http://localhost/MS Agent/msg.wav', true);
```

Затем, уже из кэша, файл проигрывается методом `Speak`:

```
Character.Speak('Обратите внимание на сообщение!', 'msg.wav');
```

Метод `Speak` мы подробно разберем в следующем разделе. Пока же замечу, что хотя в основном он используется для чтения нужного текста, с его помощью также можно проиграть звуковой файл, указанный во втором параметре метода `Get`, при этом воздушный шарик будет содержать текст, являющийся значением первого параметра.

Наконец-то мы дошли до описания последнего — третьего — параметра метода `Get` — `Queue`. Несмотря на его самый многообещающий (по объему допустимых смыслов) тип `OleVariant`, этот параметр может принимать всего два значения логического типа, т. е. `True` или `False`. Смысл параметра `Queue` заключается в том, чтобы определить необходимость постановки данного запроса в очередь всех запросов персонажа. Если параметр имеет значение `True`, то запрос ставится в очередь для текущего персонажа, и все последующие запросы ждут загрузки данных, затребованных методом `Get`. В ином случае очередь запросов персонажа продолжает выполняться, а данные отыскиваются независимо от текущих действий.

Ставить в очередь запрос `Get` следует в том случае, когда он выполняется непосредственно перед запуском загружаемой анимации. Запрашивая данные анимации, так сказать, на всякий случай, не следует тормозить выполнение очереди запросов отыскиванием анимаций.

Замечание

Перед выполнением любой анимации надо найти ее методом `Get`, иначе никакого эффекта не будет, а объект `Request` будет содержать причину данного состояния.

Пример приложения из подкаталога 15_WebAgent демонстрирует работу различных вариантов описанных операций (рис. 2.31). Проследите, пожалуйста, выполнение запрограммированных действий.

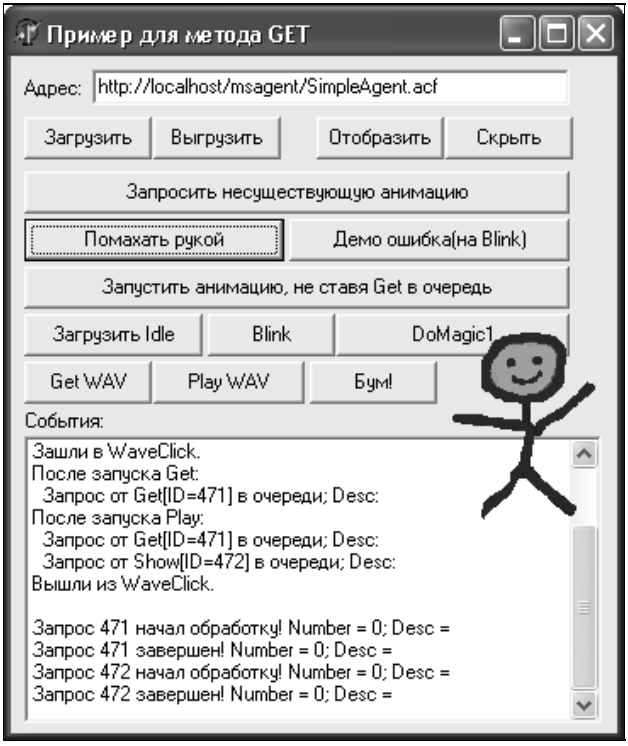


Рис. 2.31. Пример, демонстрирующий работу метода `Get` с использованием загрузки персонажа через Интернет. Хотя мой персонаж и неказист, зато сделан своими руками

В верхней части окна приложения (см. рис. 2.31) располагается поле ввода адреса персонажа. По щелчку на кнопке **Загрузить** осуществляется попытка его загрузки. Для выгрузки, скрытия и отображения также приспособлены соответствующие кнопки. При выполнении наиболее интересных действий в поле описания событий **События** (в логе событий) отображается состояние запросов внутри процедур, т. е. после запуска метода `Get` и после запуска анимации состояния уже обоих запросов. Вместе с тем лог событий отображает информацию о начале обработки запроса и о его завершении.

Щелкните на кнопке **Загрузить**, которая запускает несуществующую анимацию. В обработчике событий происходит попытка поиска методом `Get` вместе с реальной анимацией Pleasure заведомо отсутствующей анимации Idling,

однако персонаж невозмутимо начинает размахивать руками, правда, в логе мы можем заметить сообщение о том, что при выполнении запроса было что-то не так.

Персонаж поддерживает анимацию Blink (моргание глазами), и, как и для воспроизведения всех анимаций, сначала ее нужно отыскать. А вот то, что происходит при невыполнении этого указания, можно увидеть, щелкнув на кнопке **Демо ошибка**.

Если долго не обращаться к персонажу, то, как известно, он начинает бездельничать. С нашим персонажем этого не происходит, даже если очень долго ждать. Причина кроется в том, что анимации соответствующего состояния нужно загружать. Щелкните на кнопке **Загрузить Idle**, и метод `Get` отыщет на сервере анимации всех уровней бездействия, а наш самодельный персонаж, бездельничая, начнет подмигивать, моргать глазами и пинать мячик.

Не забудем и о проверке функционирования кэша. Если вы загрузили анимации состояния Idle, то анимация Blink (могание глазами) уже должна находиться в кэше (т. к. при сборке персонажа я ассоциировал ее с состоянием безделья), следовательно, запускать ее можно без предварительного отыскания методом `Get`. Проверьте это щелчком на кнопке **Blink** или **Демо ошибка**; в обработчике этих событий не включено выполнение `Get`, и персонаж должен весело моргнуть глазами без всяких ошибок.

Чтобы очистить кэш и провести эксперименты заново, воспользуйтесь диалоговым окном **Свойства обозревателя**, доступным через пункт **Свойства** контекстного меню ярлыка Internet Explorer на рабочем столе или соответствующим апплетом Панели управления. В этом диалоговом окне на вкладке **Общие** в группе элементов **Временные файлы Интернета** щелкните мышью на кнопке **Удалить файлы** и подтвердите ваши действия.

Надеюсь, наигравшись с примером и внимательно рассмотрев состояния запросов после выполнения определенных действий, читатели поймут смысл метода `Get`. Такие накладки перед проигрыванием анимации, как отыскивание данных анимации, вполне обоснованы — экономия трафика налицо.

Методы остановки запросов

Все рассмотренные до сего момента методы касались отправления запросов к персонажу — это методы проигрывания анимаций, запуска определенных состояний и даже метод отыскивания анимаций `Get`. Мы знаем, что при опережающем запуске действий создается очередь запросов. Следовательно могут возникнуть ситуации, когда необходимо либо отменить какой-либо запрос, т. е. прервать его выполнение, либо очистить очередь запросов. На этот случай у объекта `Character` также имеются все необходимые методы. Перейдем к их рассмотрению.

Всего выделить можно три подобных метода:

- `Interrupt`
- `Stop`
- `StopAll`

Метод `Interrupt` прерывает выполнение запроса другого персонажа — не того, который запускает метод. Пусть, например, приложение использует два персонажа, причем у одного из них запускается анимация и указатель на объект `Request` сохраняется в определенной переменной. Для того чтобы остановить заданное действие, другим персонажем запускается метод `Interrupt` с параметром запроса от первого персонажа. Данный метод также возвращает указатель на объект `Request`, т. е. он также ставится в очередь запросов для данного персонажа. Поэтому вы должны следить за тем, чтобы останавливаемый персонаж был загружен и чтобы существовал объект запроса.

Если запрос, останавливаемый методом `Interrupt`, находится в процессе выполнения, то он прерывается и сервер переходит к обработке следующего в очереди запроса персонажа. Если метод `Interrupt` применяется к запросу, который находится в очереди, то он удаляется из нее, а сервер, дойдя до обработки запроса, предшествовавшего удаленному запросу, после завершения станет выполнять следующий после удаленного запрос.

В основном метод `Interrupt` используется для синхронизации действий нескольких персонажей. Эту его функцию демонстрирует приложение из каталога 16_Interrupt (рабочее окно приложения приведено на рис. 2.32). Код приложения приведен в листинге 2.11.

Листинг 2.11. Код примера 16_Interrupt (секция `interface` упрощена)

```
var
  Form1: TForm1;
  Merlin, Genie: IAgentCtlCharacterEx;
  Req: IAgentCtlRequest;

implementation

procedure TForm1.Button3Click(Sender: TObject);
begin
  Merlin.Play('wave');
  Req:=Merlin.Play('process');
  Merlin.Play('DoMagic1');
  Merlin.Play('DoMagic2');
```

```
  Merlin.Play('Restpose');
end;

procedure TForm1.btInterrupt2Click(Sender: TObject);
begin
  Genie.GestureAt(Merlin.Left, Merlin.Top);
  Genie.Speak('Мерлин, перестань!', '');
  Genie.Interrupt(Req);
  Genie.Play('restpose')
end;

procedure TForm1.btInterruptClick(Sender: TObject);
begin
  Genie.Interrupt(Req);
end;
```

В приложении использованы два персонажа: Маг (Merlin) и Джинн (Genie). Для Мага создается очередь запросов из несколько анимаций — для второй по счету сохраняется объект `Request`, она-то и будет останавливаться методом `Interrupt`, запускаемым Джинном по щелчку на кнопке **Interrupt**. В обработчике события щелчка мыши второй кнопки **Interrupt(2)** прерывание не запускается сразу, а ставится в очередь запросов для персонажа Джинн и выполнется только после окончания выполнения первых двух запросов. Посмотрите, как забавно выглядит код в работе, если создать очередь для персонажа Маг, и как только он начнет выполнять анимацию Process, остановить ее щелчком на кнопке **Interrupt(2)**. Создается впечатление, что Маг действительно прекращает свои действия после замечания Джинна.

Замечание

Для успешного функционирования метода `Speak`, т. е. для того, чтобы воспроизводилась речь, следует установить пакет речевых функций Speech API и хотя бы один русский голосовой движок. Подробнее о речевых функциях в MS Agent мы поговорим в *разд. 2.8*, а также в *гл. 3*.

Метод `Interrupt` генерирует объект `Request`, однако вы не сможете прервать выполнение `Interrupt`, применяя этот же метод к другому персонажу относительно прерывающего запроса первого, т. е. следующим образом:

```
var
  Merlin, Genie: IAgentCtlCharacterEx;
  Req, Req2: IAgentCtlRequest;
...
```

```
begin
  Merlin.Play('wave');
  Req:=Merlin.Play('process');
  req2:=Genie.Interrupt(Req);
  Merlin.Interrupt(Req2);
end;
```

Рис. 2.32. Рабочее окно приложения, демонстрирующего работу метода `Interrupt`

Происходит это потому, что прежде чем Маг выполнит метод `Interrupt` запроса персонажа Джинн, тот же метод, примененный к Джинну, уже успеет остановить анимацию Process для Мага. Более того, нет сомнений, что, и, используя третий персонаж, не удастся остановить метод `Interrupt` другим таким же методом в одной процедуре (см. пример 17_Interrupt2). Конечно, можно попытаться проверить это, запуская прерывание `Interrupt` из другого потока с крохотной задержкой во времени, чтобы объект `Request` успел создаться, но запрос не успел выполниться. Однако пример носит больше академический характер, чем практический, и если в жизни у вас появится подобная ситуация, то на все воля процессора и времени — кто кого обгонит. Поэтому для остановки всех запросов, в том числе и методом `Interrupt`, применяются внеочередные методы, которые не ставятся в очередь запросов персонажа, а выполняются сразу после запуска. Их мы и рассмотрим сейчас.

Следующий метод, применяемый для остановки запросов, это метод `Stop`. Он останавливает специфический запрос персонажа, указываемый через параметр `Request`. Метод останавливает запросы только такого персонажа, которым запускается метод `Stop`, и это значит, что вам не удастся остановить запрос одного персонажа, применяя метод к другому персонажу с объектом `Request` от первого — для этих целей есть метод `Interrupt`. Тем не менее, метод `Stop` работает так же, как метод `Interrupt`, т. е. если запрос останавливается в тот момент, когда он еще находится в очереди, то он просто из нее удаляется и сервер выполняет следующее задание. В случае применения метода `Stop` к выполняемой анимации (в частном случае) она прерывается, и обработку начинает следующий запрос. Отличие состоит в том, что данный метод выполняется сразу после его запуска, т. е. он не ждет своего часа в очереди запросов персонажа, поскольку в этом нет смысла: когда очередь дойдет до метода `Stop`, наверняка останавливать будет уже нечего — все запросы уже выполнятся.

Рассмотрим такой код:

```
var
  Chars: IAgentCtlCharacterEx;
  Req: IAgentCtlRequest;
begin
  Chars:=Agent1.Characters.Character('MyAgent');
  Chars.Play('Wave');
  Req:=Chars.Play('Process');
  Chars.Play('DoMagic1');
  Chars.Play('DoMagic2');
  Chars.Stop(Req);
end;
```

В приведенном коде запускаются четыре анимации, однако в конце кода используется метод `Stop`, останавливающий второй запрос. Так как код будет выполнен очень быстро, а анимации продолжаются долго, то очередь до анимации Process не успеет дойти, и, следовательно, она будет пропущена. Конечно же, этот пример носит чисто теоретический характер, ведь совершенно ни к чему методом `Stop` в этой процедуре останавливать запрос, запущенный в ней же.

Метод `Stop` может использоваться как с параметром, так и без него. В методе `Load` для загрузки персонажа по-умолчанию мы пропускали параметр имени файла ассистента через подстановку системной переменной `EmptyParam` вместо конкретного значения. Тут нужно поступать точно так же. Метод `Stop` без параметров останавливает текущее действие персонажа,

а также очищает очередь запросов агента; кроме того, останавливаются все очередные запросы вызова методов `Get`[6] и `Interrupt`.

Подобный вариант вызова метода `Stop` обычно применяют, когда необходимо заставить персонаж обратить внимание на пользователя. Ведь самое естественное действие человека с целью остановки агента, свойственное даже неподготовленным пользователям, — это щелканье на нем мышью. Поэтому в обработчике события щелчка мыши обычно прописывают следующий код:

```
procedure TForm1.Agent1Click(Sender: TObject;
  const CharacterID: WideString; Button, Shift, x, y: Smallint);
begin
 Agent1.Characters.Character(CharacterID).Stop(EmptyParam);
end;
```

Для демонстрации функционирования метода `Stop` подготовлен проект каталога 18_StopMeth, в котором показаны различные варианты использования этой функции. В следующем примере 19_StopMeth2 показана остановка метода `Interrupt` методом `Stop` в различных вариантах:

```
req:=Genie.Play('Processing');
Merlin.MoveTo(100, 100, 1000);
Merlin.Play('Wave');
r:=Merlin.Interrupt(req);
Merlin.GestureAt(600, 600);
Merlin.Play('Decline');
Merlin.Stop(r);
```

И другой вариант без параметров в `Stop`:

```
req:=Genie.Play('Processing');
Merlin.MoveTo(100, 100, 1000);
Merlin.Play('Wave');
Merlin.Interrupt(req);
Merlin.GestureAt(600, 600);
Merlin.Play('Decline');
Merlin.Stop(EmptyParam);
```

В обоих случаях метод `Stop` прерывает выполнение метода `Interrupt` и персонаж Джинн продолжает выполнять анимацию без остановки.

И наконец последний метод, управляющий остановкой запросов, — метод `StopAll`. По названию можно предположить, что метод управляет прерыва-

[6] Только очередные запросы метода `Get`, т. е. те, которые ставятся в очередь персонажа. Параметр `Queue` запросов `Get`, ставящихся в очередь, равен `True`.

нием нескольких запросов. Так оно и есть. Через его единственный параметр можно указывать, какую группу или группы запросов мы желаем остановить:

```
procedure StopAll(Types: OleVariant);
```

Возможные значения единственного параметра метода `StopAll` приведены в табл. 2.8.

Таблица 2.8. *Значения параметра метода* `StopAll`

Значение	Описание
`Get`	Все вызовы метода `Get` в очереди персонажа
`NonQueuedGet`	Все неочередные запросы, вызванные методом `Get`
`Move`	Все запросы на перемещение в очереди агента
`Play`	Все методы проигрывания анимации. Не только через метод `Play`, сюда входят и запросы метода `GestureAt`
`Speak`	Все "говорящие" методы, т. е. `Speak` и `Think`

Так же, как и в методе `Get`, необходимые значения можно комбинировать, указывая их через запятую, например:

```
Character.StopAll('Play, Move');
```

Демонстрируют метод `StopAll` два примера приложений из каталогов 20_StopAllMeth и 21_StopAllMeth2. Первое приложение (его рабочее окно показано на рис. 2.33) построено с использованием персонажа из раздельного формата (нашего самодельного агента), чтобы продемонстрировать остановку очередных и неочередных запросов метода `Get`, т. е. для проверки примера потребуется использовать web-сервер, как мы это делали ранее.

Все значения параметра `Type` метода `StopAll`, кроме `NonQueuedGet`, можно проверить, создавая очередь запросов, а затем останавливая группы действий кнопкой **StopAll**, предварительно выбрав значение параметра `Type` в раскрывающемся списке рядом с кнопкой **StopAll** (на рис. 2.33 показан уже раскрытый список).

Учитывая, что код процедуры выполняется очень бытро, невозможно поймать тот момент, когда последствия остановки запросов типа `NonQueuedGet` будут очевидны. Для того чтобы все-таки прочувствовать это, метод `StopAll` со значением параметра `NonQueuedGet` применяется прямо в коде обработчика события, если установлен флажок **Остановить в коде запросы NonQueuedGet**.

Второй проект демонстрирует работу метода `StopAll`, используя обычный формат персонажа Merlin. Пример немного отличается от предыдущего. Од-

нако в состав очереди запросов входят такие методы, как `GestureAt`, которые обрываются методом `StopAll` с значением параметра, равным `Play`, и метод `Interrupt`, прерывающийся запуском `StopAll` с любым значением параметра, из-за чего второй используемый персонаж Genie продолжает выполнять анимацию.

Рис. 2.33. Рабочее окно приложения 20_StopAllMeth, демонстрирующего работу метода `StopAll`

Обобщим сведения о методах остановки запросов.

Метод `Interrupt` служит для остановки запросов другого персонажа, поэтому используется для синхронизации действий персонажей. Он является очередным запросом, т. е. ставится в очередь запросов агента, и поэтому его можно отменять другими более функциональными методами.

Методы `Stop` и `StopAll` выполняются сразу после их запуска и не ставятся в очередь, т. е. имеют равную силу. Они могут прервать метод `Interrupt` и, естественно, другие запросы. Методу `Stop` через объект `Request` можно указывать, какой именно запрос необходимо остановить; без параметров (т. е. с указанием переменной `EmptyParam`) метод останавливает текущее действие и очищает очередь запросов, в том числе и все вызовы `Get` в очереди. Метод `StopAll` наиболее универсален и имеет самую широкую сферу влияния на запросы. Он может остановить как методы анимаций, так и очередные и внеочередные вызовы метода `Get`.

Галерея Default Character Properties — своими руками

Мы уже рассматривали *(см. разд. 2.6)* применение метода `ShowDefaultCharacterProperties`, отображающего диалоговое окно своеобразной галереи — Default Character Properties (Свойства персонажа по умолчанию), некорректно работающее на некоторых машинах. Поэтому предлагаю создть подобную функцию своими руками. Этот опыт послужит для обобщения некоторых новых знаний о методах и свойствах MS Agent.

В окне галереи (см. рис. 2.12) мы видим изображение персонажа, его имя и описание, кнопки прокрутки видов агентов, а также вспомогательные надписи. Сделаем примерно то же самое.

Не буду подробно описывать код, обозначу лишь главные моменты. При открытии окна выбора агентов наш персонаж подлетает в назначенное место. Во время прокрутки видов возможных агентов персонаж меняет облик, однако при щелчке на кнопке **Отмена** текущий персонаж заменяется на прежний. По щелчку на кнопках **Применить** или **OK** выбранный вид агента запоминается, кроме того, в процедуре обработки события щелчка мышью на кнопке **OK** диалоговое окно выбора персонажа закрывается.

Посмотрим на скриншот окна выбора видов агентов Default Character Properties, работа с которым реализована в проекте 22_DefChar (рис. 2.34), и обратим внимание на то, какие сведения о персонаже Merlin отображаются в нем.

Если в приложение необходимо вставить выбор персонажей через собственноручно созданное окно **Default Character Properties**, то к проекту необходимо подлючить модуль DefCharForm.pas, положить в папку программы DFM-файл формы и в нужном месте вызвать нашу функцию `ShowDefaultCharacterProperties`.

В связи с тем, что созданная нами функция не является методом какого-либо объекта или интерфейса, ей необходимы некоторые параметры. Возможно, если бы кто-то из читателей реализовывал бы подобную функцию,

он выбрал бы другой подход. Однако автор решил использовать три параметра: идентификатор персонажа (`CharacterID`), указатель на интерфейс (`IAgentCtlCharacters`) и указатель на переменную, в которой хранится исходное имя файла текущего персонажа в приложении. Прототип функции приведен ниже:

```
procedure ShowDefaultCharacterProperties(CharacterID: WideString;
     Characters: IAgentCtlCharacters; CharFileName: Pointer);
```

Рис. 2.34. Диалоговое окно **Default Character Properties**, реализованное совими руками

В виде идентификатора персонажа необходимо передавать `CharacterID` того агента, которого вы собираетесь менять. Параметр `Characters` — это указатель на интерфейс коллекции персонажей, и передавать его нужно для того, чтобы функция могла перезагружать персонажи в коллекции. Конечно, можно передавать указатель и на элемент управления Agent Control (интерфейс `IAgentCtlEx`), так что со своей стороны могу только предложить вам переделать модуль по собственному усмотрению. Третий параметр —

`CharFileName` — представляет собой указатель на переменную, хранящую имя файла агента, которого следует менять; параметр, присутствует для того, чтобы автоматически в случае нажатия кнопки **Отмена** вернуть предыдущий персонаж. Необходим именно указатель на переменную, потому что функция записывает в переменную по этому адресу новое имя файла агента, чтобы главная программа узнала о смене. Это удобно, если в программе вы используете сохранение имени файла текущего агента.

Итак, задача реализации "собственноручной" функции `ShowDefaultCharacterProperties` решена. Созданная функция отображает аналогичное окно выбора персонажа из имеющегося набора, показывает его необходимые свойства (и даже в большем количестве, чем это делает стандартное диалоговое окно), а также заменяет агента и извещает главное приложение об изменившемся персонаже при помощи адреса переменной с именем файла текущего агента.

Получаем список анимаций

Мы уже достаточно полно рассмотрели различные методы MS Agent, однако до сих пор не касались работы со списком анимаций, т. е. со структурой `AnimationNames`. Сейчас самое время этим заняться.

Из схемы объектной модели MS Agent (см. рис. 2.13) видно, что доступ к коллекции анимаций персонажа лежит через объект `Character`. Это естественно, ведь набор анимаций индивидуален для каждого персонажа. Обратиться к набору анимаций можно следующим образом:

```
(Agent1.Characters.Character('MyAgent') as
                    IagentCtlCharacterEx).AnimationNames
```

Коллекция `AnimationNames` расширенного интерфейса объекта `Character` интерфейса `IAgentCtlCharacterEx` содержит список имен всех анимаций для данного персонажа, если используется его структурный формат, т. е. ACS-файл, или список доступных анимаций для раздельного формата ACF (имеются в виду только те анимации, которые были запрошены методом `Get`). После прочтения этого определения сразу представляется набор строк типа `TStrings`, однако доступ к списку анимаций намного сложнее. Интерфейс `IAgentCtlAnimationNames`, указателем на который является свойство `AnimationNames`, обладает единственным свойством `Enum` (от Enumerator — перечислитель), посредством которого можно "пробегать" по всем анимациям и узнавать их имена.

```
IAgentCtlAnimationNames = interface(IDispatch)
  ['{8B77181C-D3EF-11D1-8500-00C04FA34A14}']
  function  Get_Enum: IUnknown; safecall;
  property Enum: IUnknown read Get_Enum;
end;
```

Для работы с разными перечислителями в Delphi существует несколько типов интерфейсов: для строк это IEnumString, для интерфейсов это IEnumUnknown, для типа Variant — IEnumVariant. Учитывая, что свойство Enum и метод Get_Enum возвращают указатель на интерфейс IUnknown, нужно запросить интерфейс, необходимый нам для перечисления элементов списка. Несмотря на всю очевидность выбора IEnumString, опытным путем установлено, что интерфейс IEnumString не поддерживается интерфейсом IAgentCtlAnimationNames, что объясняет использование более универсального интерфейса IEnumVariant.

```
IEnumVariant = interface(IUnknown)
  ['{00020404-0000-0000-C000-000000000046}']
  function Next(celt: LongWord; var rgvar : OleVariant;
    out pceltFetched: LongWord): HResult; stdcall;
  function Skip(celt: LongWord): HResult; stdcall;
  function Reset: HResult; stdcall;
  function Clone(out Enum: IEnumVariant): HResult; stdcall;
end;
```

Замечание

Читатель может спросить: что нужно было бы ему делать, если бы он самостоятельно изучал работу с MS Agent? Как "дойти" до мысли о том, что требуется использовать интерфейс IEnumVariant, что именно его следует запрашивать у свойства Enum? Да никак! Изначально технология Microsoft Agent ориентирована на VBasic и VC++, т. е. на "любимые" языки Microsoft, поэтому в Delphi не так просто заставить работать некоторые моменты. Тем более что в англоязычной документации приведены примеры только на Visual Basic и немного на VC++. Вот и приходится ориентироваться на свой жизненый опыт и учитывать особенности программирования в разных средах. Пример для Basic со списком анимаций в документации к MS Agent сходу можно очень неоднозначно интерпретировать для Delphi:

```
For Each Animation in Genie.AnimationNames
  Genie.Play Animation
Next
```

Приведенный код на VB демонстрирует проигрывание всех анимаций по порядку для данного персонажа.

Итак, для того чтобы узнать список анимаций, мы обращаемся к свойству Enum коллекции AnimationNames и запрашиваем у него интерфейс IEnumVariant, т. е. пишем следующий код:

```
var
  Chars: IAgentCtlCharacterEx;
  AEnum: IEnumVariant;
```

```
Begin
  Agent1.Characters.Load('MyAgent', 'merlin.acs');
  Chars:=Agent1.Characters.Character('MyAgent');
  Chars.AnimationNames.Enum.QueryInterface(IEnumVariant, AEnum);
  ...
end;
```

Необязательно использовать метод `QueryInterface`, вполне подойдет оператор `as`, как для для случаев приведения переменной к нужному типу:

```
AEnum:=Chars.AnimationNames.Enum as IEnumVariant;
```

На самом деле это далеко не все. Как говорится, "стрижка только началась!" Однако теперь многое более или менее очевидно. Интерфейс `IEnumVariant` имеет все необходимые методы для перебора элементов в перечислении (табл. 2.9).

Таблица 2.9. *Методы интерфейса* `IEnumVariant`

Метод	**Описание**
`function Next (celt: LongWord; var rgvar : OleVariant; out pceltFetched: LongWord): HResult; stdcall;`	Метод перемещает текущую позицию курсора в перечислении на число, равное `celt`, при этом значение нового текущего элемента списка оказывается в параметре `rgvar`, а последний параметр `pceltFetched` определяет конец списка (если он равен нулю, то текущий элемент последний, иначе за ним следуют еще элементы)
`function Skip (celt: LongWord): HResult; stdcall;`	Метод перемещает курсор в перечислении на заданное число
`function Reset: HResult; stdcall;`	Курсор сбрасывается на первый элемент списка
`function Clone (out Enum: IEnumVariant): HResult; stdcall;`	Метод копирует текущее перечисление в переменную `Enum`

Прежде всего нужно сбросить курсор на первый элемент списка, а затем мы будем постепенно перемещать его по перечислению и попутно каждое имя анимации выводить в компонент `ListBox`. Как только флаг `pceltFetched` метода `Next` будет равен нулю, мы прекратим действия:

```
var
  Chars: IAgentCtlCharacterEx;
  AEnum: IEnumVariant;
```

```
  flag: Cardinal;
  V: OleVariant;
begin
  Agent1.Characters.Load('MyAgent', 'merlin.acs');
  Chars:=Agent1.Characters.Character('MyAgent') as IAgentCtlCharacterEx;
  Chars.AnimationNames.Enum.QueryInterface(IEnumVariant, AEnum);
  {Получаем интерфейс анимаций агента:}
  AEnum.Reset; {Сбрасываем курсор на первый элемент списка анимаций}
  ListBox1.Items.Clear;
  repeat
    AEnum.Next(1, V, flag);
    if VarToStr(V) <> '' Then
      ListBox1.Items.Add(V);
  until flag=0; {Флажок будет сброшен на ноль, когда мы узнаем имя
последней анимации}
end;
```

Теперь осталось прописать код воспроизведения анимации по щелчку на ее названии в списке `ListBox`, но перед этим надо не забыть остановить действия персонажа:

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
  Chars.Stop(EmptyParam);
  Chars.Play(ListBox1.Items.Strings[ListBox1.ItemIndex])
end;
```

Полностью код можно просмотреть в проекте каталога 23_GetAnim. На его основе написан еще один пример из каталога 24_GetAnim2, являющийся по своей сути маленьким просмотрщиком анимаций всех доступных персонажей (рис. 2.35). В проекте реализован просмотр персонажей только в директории MS Agent\chars, однако ничего не мешает загрузить персонаж из файла.

Некоторые читатели спросят: "А что делать, если нужно просто узнать, поддерживается ли определенная анимация?" Что ж, придется "пробежать" по всем элементам и поискать, есть ли в списке нужная вам анимация. Конечно, кто-нибудь скажет, что это неудобно, — но как бы вы поступили в ситуации прямого поиска элемента? Для интереса взгляните, как в классе `TStrings` реализован метод `IndexOf`.

```
function TStrings.IndexOf(const S: string): Integer;
begin
  for Result := 0 to GetCount - 1 do
```

```
    if AnsiCompareText(Get(Result), S) = 0 then Exit;
  Result := -1;
end;
```

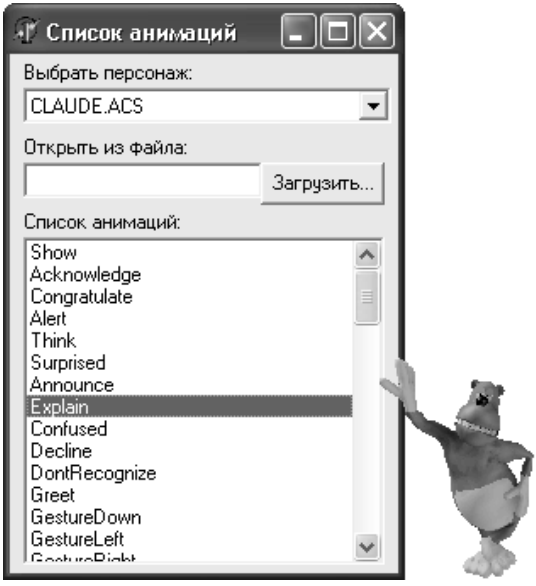


Рис. 2.35. Рабочее окно просмотрщика анимаций. На рисунке персонаж Claude

Любой из вас может реализовать подобную функцию, тем более что обычно количество анимаций редко доходит до сотни, и поэтому даже самый простейший алгоритм поиска будет работать более или менее сносно.

Раскрывающееся меню и объект *Commands*

За время обследования MS Agent вы, наверное, часто пользовались контекстным меню персонажа для его скрытия или отображения. Кроме того, ранее *(в разд. 2.1)* было замечено, что агент может влючать в список доступных команд глобальные команды, такие как **Скрыть** (Hide) и **Открыть окно команд голосом** (Open The Voice Commands Window), а также команды текущего и других приложений. А сейчас мы поговорим о добавлении в меню наших собственных команд.

Список команд определенного персонажа хранит специальная коллекция `Commands` (интерфейсы `IAgentCtlCommands` и `IAgentCtlCommandsEx`), каждый элемент которой — `Command` — является отдельной командой.

```
(Agent1.Characters.Character('MyAgent') as
                IagentCtlCharacterEx).Commands.Command('CommandName')
```

Коллекция `Commands` обладает следующими методами для оперирования командами:

- `Add`
- `Command`
- `Insert`
- `Remove`
- `RemoveAll`

Описание методов приведено в табл. 2.10.

Таблица 2.10. *Методы объекта* `CommandsObject`

Метод	Описание
`function Add (const Name: WideString; Caption: OleVariant; Voice: OleVariant; Enabled: OleVariant; Visible: OleVariant): IAgentCtlCommand; safecall;`	Добавляет в коллекцию новую команду с именем `Name`, подписью в меню `Caption` и голосовой фразой для распознавания `Voice` (этот параметр мы не будем рассматривать в данном разделе, подробнее *см. разд. 2.8*). С помощью параметра `Enabled` указывают, будет ли команда доступна в меню (`True` — да, `False` — нет, в этом случае будет наблюдаться характерное затемнение текста); параметр `Visible` также показывает видимость команды в меню (`True` — видима, `False` — скрыта)
`function Command (const Name: WideString): IAgentCtlCommandEx; safecall;`	Возвращает указатель на интерфейс конкретной команды по ее имени
`function Insert (const Name: WideString; const RefName: WideString; Before: OleVariant; Caption: OleVariant; Voice: OleVariant; Enabled: OleVariant; Visible: OleVariant): IAgentCtlCommand; safecall;`	Вставляет команду с именем `Name`, перед командой с именем `RefName`, если параметр `Before` установлен в значение `True`, и после команды с именем `RefName`, если параметр `Before` принимает значение `False`. Остальные параметры аналогичны соответствующим параметрам из метода `Add`
`procedure Remove (const Name: WideString); safecall;`	Удаляет команду с именем `Name` из коллекции
`procedure RemoveAll; safecall;`	Очищает коллекцию от всех команд

Объект `Commands` имеет несколько свойств, однако мы не будем сейчас рассматривать их все, а разберем лишь некоторые, остальных же коснемся в

следующих разделах. Полный список свойств объекта `Commands` приведен в табл. 2.11.

Таблица 2.11. *Свойства объекта* `Commands`

Свойство	**Описание**
Свойства интерфейса `IAgentCtlCommands`	
`property Caption: WideString`	Подпись данной коллекции персонажа
`property Count: Integer`	Количество команд
`property Voice: WideString`	Определяет фразу для распознавания, которая будет означать команду переключения между группами команд *(см. разд. 2.8)*
`property Enum: IUnknown`	Перечислитель всех команд
`property Visible: WordBool`	Определяет видимость пункта меню для переключения между разными группами коллекций команд, если персонаж используют несколько приложений *(см. разд. 2.8)*
`property Item[const Name: WideString]: IAgentCtlCommandEx`	Список команд
Свойства интерфейса `IAgentCtlCommandsEx`	
`property DefaultCommand: WideString`	Команда по умолчанию (выделяется полужирным шрифтом), ни на что не влияет, в том числе и на выполнение команды; представляет собой элемент условности (т. е. "для красоты")
`property FontName: WideString`	Имя шрифта начертания команд в раскрывающемся меню
`property FontSize: Integer`	Размер шрифта в раскрывающемся меню
`property GlobalVoiceCommandsEnabled: WordBool`	Определяет, будут ли доступны такие глобальные голосовые команды персонажа, как **Показать**/**Скрыть** и **Открыть**/**Закрыть окно команд голосом**. Данное свойство не влияет на отображение этих команд в раскрывающемся меню, однако влияет на их распознавание не только в данном активном клиент-приложении, но и во всех остальных *(см. разд. 2.8)*
`property HelpContextID: Integer`	Определяет идентификатор контекстной справки *(подробнее см. следующий подраздел данного раздела)*

Таблица 2.11 *(окончание)*

Свойство	Описание
Свойства интерфейса `IAgentCtlCommandsEx`	
`property VoiceCaption: WideString`	Определяет заголовок голосовой фразы, которая будет означать команду переключения между группами команд *(см. разд. 2.8)*

Как уже отмечалось, в данном подразделе мы рассмотрим только часть свойств; также мы опустим здесь описание ряда параметров некоторых методов.

Мы знаем, что контекстное меню состоит из трех частей: глобальной неизменяемой части, куда входят команды **Отображение/Скрытие** (Show/Hide) и **Открыть/Закрыть окно команд голосом** (Open/Close Voice Commands Window); команды текущего активного приложения и команды других приложений. Так вот, команды глобальной части невозможно изменять, они присутсвуют всегда. Однако странно, что раньше мы не наблюдали в списке команд пункта **Открыть окно команд голосом**. Это происходило из-за того, что наше приложение не обладало ни одной командой, тем более голосовой. Как только мы добавим хотя бы одну команду, данный пункт появится. Не все команды могут быть голосовыми, а что потребуется сделать, чтобы добавить команду, распознаваемую голосом, мы узнаем из *разд. 2.8*, пока же разберемся с тем, как использовать контекстное меню для своих целей.

Из табл. 2.10 видно, что метод `Add` объекта `Commands` добавляет команду в коллекцию, например:

```
var
  Chars: IAgentCtlCharacterEx;
begin
  Agent1.Characters.Load('MyAgent', 'merlin.acs');
  Chars:=Agent1.Characters.Character('MyAgent') as IAgentCtlCharacterEx;
  Chars.Commands.Add('Move', 'В центр экрана', '', true, true);
  Chars.Commands.Add('Play', 'Мотор!', '', true, true);
end;
```

Здесь к персонажу `Chars` добавляется одна команда с именем `Move`, которая будет видима и доступна в контекстном меню под названием **В центр экрана**, и другая — с именем `Play` и подписью в меню **Мотор!** — тоже видимая и доступная.

Замечание

Если при добавлении команды не указать ее подпись, то будет тот же эффект, что и при установлении свойства `Visible` в значение `False`.

Все действия по написанию кодов обработчиков события щелчка мышью описываются в процедуре обработки события `OnCommand`. Там в виде параметра передается указатель на интерфейс `IDispatch`, через который можно получить доступ к интерфейсу ввода пользователя `IAgentCtlUserInput`, узнать имя полученной команды и другие ее параметры:

```
procedure TForm1.Agent1Command(Sender: TObject;
  const UserInput: IDispatch);
var
 cmd: IAgentCtlUserInput;
begin
 UserInput.QueryInterface(IID_IAgentCtlUserInput, cmd);
 if cmd.Name='Move' then
  begin
   Chars.MoveTo((Screen.Width div 2)-(Chars.Width div 2),
               (Screen.Height div 2)-(Chars.Height div 2), 1000);
  end;
 if cmd.Name='Play' then
  begin
   Chars.Play('DoMAgic1');
   Chars.Play('DoMagic2');
   Chars.Play('restpose');
  end;
end;
```

Как видите, все просто: получаем указатель на интерфейс через метод `QueryInterface`, затем сравниваем имя команды `Name` с заданным значением и воспроизводим нужные действия.

Нередко бывают случаи, когда некоторые пункты меню не нужны или временно не должны быть доступны. Управляющие этим параметры изменяются через свойства конкретной команды — объекта `Command` (интерфейсы `IAgentCtlCommand` и `IAgentCtlCommandEx`) коллекции `Commands`:

```
Chars.Commands.Command('Play').Enabled:=false;
Chars.Commands.Command('Move').Visible:=false;
```

Данный код делает команду `Play` недоступной, т. е. она затемняется, а пункт меню команды `Move` скрывается. Полный список свойств объекта `Command` приведен в табл. 2.12.

Таблица 2.12. *Свойства объекта* `Command`

Свойство	Описание
Свойства интерфейса `IAgentCtlCommand`	
`property Caption: WideString`	Определяет подпись команды в контекстном меню
`property Confidence: Integer`	Число, определяющее достоверность распознанной команды. Если значение этого свойства меньше значения свойства `UserInput.Confidence` в событии `OnCommand`, то в подсказке `Listening Tip` отображается значение свойства `Confidence` *(см. разд. 2.8)*
`property ConfidenceText: WideString`	Текст, выводимый в окне подсказки `Listening Tip`, если значение свойства команды `Confidence` меньше значения свойства `UserInput.Confidence` в событии `OnCommand` *(см. разд. 2.8)*
`property Enabled: WordBool`	Определяет доступность команды. Если команда определена как недоступная, то она также недоступна и для речевого ввода
`property Visible: WordBool`	Определяет видимость пункта меню, соответствующего данной команде. Свойство не влияет на отображение команды в окне **Команды голосом** (Voice commands) и ее распознавание
`property Voice: WideString`	Голосовая фраза, вызывающая данную команду *(см. разд. 2.8)*
Свойства интерфейса `IagentCtlCommandEx`	
`property HelpContextID: Integer`	Определяет идентификатор справочной службы *(см. следующий подраздел)*
`property VoiceCaption: WideString`	Подпись голосовой команды в окне **Команды голосом** (Voice commands) *(см. разд. 2.8)*

Контекстное меню раскрывается автоматически при нажатии правой кнопки мыши, однако эта опция также является настраиваемой, что необходимо в тех случаях, когда требуется раскрыть меню по нажатию другой клавиши или отобразить собственное меню. Свойство `AutoPopupMenu` объекта `Character` интерфейса `IAgentCtlCharacterEx` определяет, должно ли контекстное меню автоматически появляться по щелчку правой кнопкой мыши на персонаже (если его значение равно `True`, то меню появляется самостоятельно, иначе — нет).

Например:

```
Character.AutoPopupMenu:=false;
```

Этим оператором мы отключили отображение меню, и теперь в обработчике события `OnClick` можно заставить меню появляться по нажатию любой кнопки мыши, даже по левой или средней:

```
procedure TForm1.Agent1Click(Sender: TObject;
  const CharacterID: WideString; Button, Shift, x, y: Smallint);
begin
 if(not Character.AutoPopupMenu) and(Button = 1) then
   Character.ShowPopupMenu(x, y);
end;
```

Демонстрационный пример из каталога 25_PopupMenu (рабочее окно приложения приведено на рис. 2.36) показывает функционирование рассмотренных в данном подразделе методов и свойств.

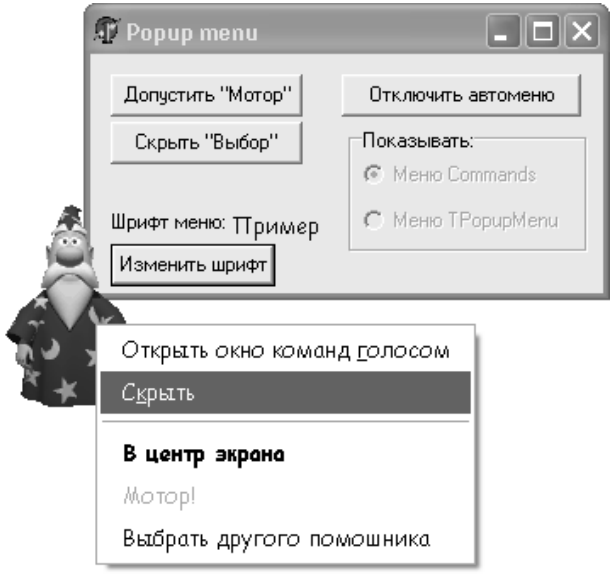


Рис. 2.36. Рабочее окно **Popup menu** приложения, демонстрирующего работу раскрывающегося меню команд персонажа

Следует отметить, что свойство `AutoPopupMenu` не влияет на другие персонажи вашего приложения, а также на остальные клиентские приложения, использующие тот же самый персонаж, т. е. эффект появляется только в том приложении, которое изменяет данное свойство.

Персонаж и справочная служба

Быть может, одна из самых главных функций анимированного персонажа — это функция помощи пользователю, в том числе и интеллектуальной. Естественно, в MS Agent не реализованы какие-нибудь сложные алгоритмы искусственного интеллекта, однако некоторые методы для связи персонажа с обычной справочной службой существуют. Их немного, поэтому данный подраздел будет коротким.

Первый делом необходимо создать справочный файл формата WindowsHelp, т. е. файл с расширением hlp. Для создания справки можно, конечно, воспользоваться программой Microsoft Windows Help Compiler, однако в ходе создания файла помощи к демонстрационному примеру автор использовал пакет Microsoft Help Workshop, входящий в поставку Delphi и доступный в адресу $(Delphi)\Help\Tools\hcw.exe, где $(Delphi) — путь к каталогу с установленной средой.

Не буду подробно объяснять технологию разработки справочного файла, т. к. эта тема выходит за рамки данной книги. Поясню лишь самые основные моменты. Создание справки обычно начинается с написания ее текста в каком-либо текстовом редакторе, например, MS Word, и с сохранения его (как правило) в файле формата RTF. В файле расставляются ключевые слова, ссылки перехода между разделами, а также указатели разделов справки. Каждый раздел начинается с новой страницы, т. е. после знака разрыва страницы. Начало раздела помечается сноской типа #, в которой указывается идентификатор раздела, обычно составленный из любых букв (но лучше всего начинать название идентификатора с префикса `IDH_` — тогда компилятор справки проверит ссылки на разделы). Затем подготавливается непосредственно файл справки в какой-либо программе, например, в Windows Help Compiler. В ней указывается исходный RTF-файл, определяются ссылки разделов, дополнительные параметры положения окна и его размеры по умолчанию, и, наконец, проект компилируется. При удачной компиляции создается соответствующий файл с расширением hlp. Для примера мной подготовлен файл справки, к которому прилагаются исходные ресурсы. Они находятся в том же каталоге 26_AgentHelp, что и демонстрационный проект.

Связь со справочной службой персонажа осуществляется следующим образом. Для конкретного персонажа в приложении определяется справочный файл с помощью свойства `HelpFile`, затем через свойство `HelpContextID` объекта `Character` (интерфейса `IAgentCtlCharacterEx`) или одноименного свойства любой команды коллекции `Commands` определяется идентификатор раздела справки, и наконец, включается справочный режим через свойство `HelpModeOn`.

```
Chars.HelpFile:=ExtractFilePath(ParamStr(0))+'DemoHelp.hlp';
Chars.HelpContextID:=2;
```

```
Chars.Commands.Add('Play', 'Мотор!', '', true, true);
Chars.Commands.Add('Help', 'Справка...', '', true, true);
(Chars.Commands.Command('Help') as IAgentCtlCommandEx).HelpContextID:=3;
Chars.HelpModeOn:=true;
```

В заданном с помощью приведенного выше кода режиме работы со справкой указатель мышки при наведении ее на персонаж превращается в характерный указатель с вопросительным знаком (рис. 2.37). По щелчку мыши на персонаже в дополнительном окне открывается раздел справки, указанный в свойстве `HelpContextID`, то же самое происходит при выборе команды из списка меню. Если в свойстве `HelpContextID` ничего не было указано, или если не определен справочный файл, то будет выдано сообщение о том, что "с элементом не сопоставлен ни один раздел справки". Кроме того, при выборе глобальных команд MS Agent будет отображать собственные разделы справки.

Рис. 2.37. В режиме справки указатель мыши при наведении на персонаж принимает вид стрелки с вопросительным знаком

Режим справки отключается, когда пользователь щелкает на каком-либо элементе управления MS Agent: на самом персонаже, элементе меню и т. п. При этом, если элементу сопоставлен раздел справки, последний открывается (см. рис. 2.38), одновременно генерируется событие `OnHelpComplete` с параметрами, включающими: идентификатор агента `CharacterID`, имя команды `Name` (если пользователь вышел при этом из режима справки по выбору команды) и параметр причины `Cause`, ряд возможных значений которого приведен в табл. 2.13.

```
type TAgentHelpComplete = procedure(Sender: TObject; const CharacterID:
WideString; const Name: WideString; Cause: Smallint) of object;
property OnHelpComplete: TAgentHelpComplete
```

Таблица 2.13. *Описания значений параметра* `Cause`

Значение	Описание
1	Пользователь выбрал команду вашего приложения
2	Пользователь выбрал объект `Commands` другого клиент-приложения
3	Пользователь выбрал команду **Открыть голосовые команды** (Open Voice Commands)
4	Пользователь выбрал команду **Закрыть голосовые команды** (Close Voice Commands)
5	Выбрали команду **Показать** (Show)
6	Выбрали команду **Скрыть** (Hide)
7	Пользователь выбрал персонаж (щелкнул на нем)

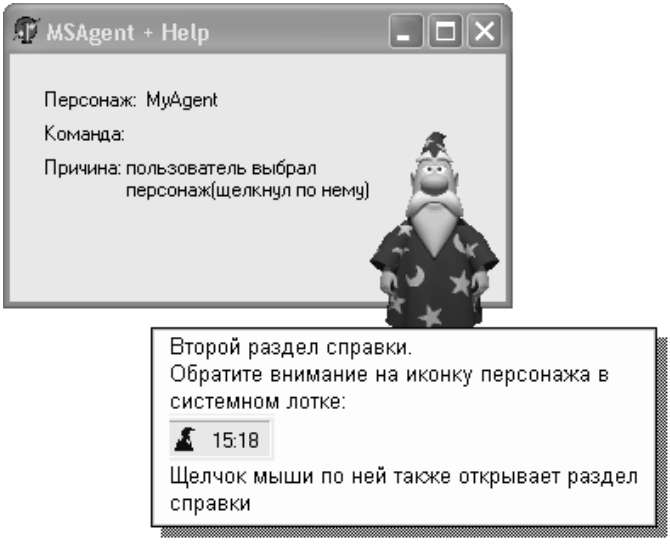


Рис. 2.38. Пример отображения раздела справки средствами MS Agent

Проект 26_AgentHelp демонстрирует работу MS Agent со справочной службой.

"Воздушный шарик" персонажа (Balloon)

Воздушный шарик с текстом речи персонажа, позаимствованный из комиксов, как нельзя лучше гармонирует с анимированным персонажем. Мы уже неоднократно видели, как он появляется и отображает читаемый методом

`Speak` и "думаемый" методом `Think` текст, однако до сих пор не обращали внимание на его параметры и не управляли им. Однажды мы заметили, что в глобальных настройках персонажа существует опция, позволяющая управлять отображением текстового воздушного шарика. Это глобальная команда, и сейчас мы рассмотрим свойства объекта `Balloon` (интерфейсы `IAgentCtlBalloon` и `IAgentCtlBalloonEx`) конкретного персонажа.

Сразу хотелось бы отметить, что воздушный шарик — объект `Balloon` — не обладает разнообразными функциональными возможностями, наподобие тех, что имеются в аналогичном объекте MS Office, однако имеется ряд сторонних разработок, более "продвинутых" в этом направлении, которые мы рассмотрим в *гл. 4*. К тому же большинство свойств объекта `Balloon` доступны только для чтения, потому что почти все они задаются при создании персонажа в редакторе персонажей Microsoft Agent Character Editor.

Чтобы не рассеивать внимание читателя, приведу таблицу (табл. 2.14) с описанием свойств объекта `Balloon` и небольшой пример из каталога 27_Balloon, демонстрирующий все изменяемые свойства текстового воздушного шарика персонажа. Рабочее окно приложения примера показано на рис. 2.39.

Таблица 2.14. *Свойства объекта* `Balloon`

Свойство	**Описание**
Свойства интерфейса `IAgentCtlBalloon`	
`property BackColor: Integer`	Возвращает значение цвета фона `Balloon`
`property BorderColor: Integer`	Возвращает цвет бордюра
`property CharsPerLine: Integer`	Возвращает количество символов в строке
`property Enabled: WordBool`	Возвращает значение, показывающее, доступен ли для персонажа объект `Balloon`. Отражает состояние соответствующей опции в окне **Дополнительные параметры персонажей** (Advanced Character Options) на вкладке **Вывод** (Output). `True` — да, иначе — нет
`property FontBold: WordBool`	Показывает, является ли стиль шрифта текста в воздушном шарике полужирным
`property FontCharSet: Smallint`	Управляет набором символов для отображения текста в воздушном шарике. Данное свойство не должно нас волновать, т. к. для английского и русского текста здесь ничего менять не надо

Таблица 2.14 *(продолжение)*

Свойство	Описание
Свойства интерфейса `IAgentCtlBalloon`	
	Большинство значений этого свойства имеют смысл для китайской, корейской и японской версий Windows. Неполный список значений: 0 — Standard Windows characters (ANSI); 1 — набор символов по умолчанию; 2 — набор символов symbol; 255 — расширенный набор символов, отображаемый MS-DOS-приложениями. Данное свойство доступно для изменения
`property FontItalic: WordBool`	Показывает, является ли стиль шрифта текста шарика курсивным
`property FontName: WideString`	Возвращает и изменяет название шрифта выводимого в воздушный шарик текста
`property FontSize: Integer`	Возвращает и изменяет размер шрифта выводимого текста
`property FontStrikethru: WordBool`	Показывает, является ли шрифт текста шарика перечеркнутым
`property FontUnderline: WordBool`	Показывается, является ли шрифт подчеркнутым
`property ForeColor: Integer`	Возвращает цвет текста
`property NumberOfLines: Integer`	Возвращает число строк текста
`property Visible: WordBool`	Позволяет управлять видимостью объекта `Balloon`. Значение свойства автоматически изменяется, когда текстовый шарик отображается или скрывается при вызове методов `Speak` и `Think`. Ручное изменение свойства имеет эффект, если вызовы методов `Speak` и `Think` не выполняются и не стоят в очереди — например, по завершении работы метода `Speak` и исчезновении `Balloon` его можно вновь отобразить с предыдущим текстом

Таблица 2.14 *(окончание)*

Свойство	Описание
Свойства интерфейса `IagentCtlBalloonEx`	
`property Style: Integer`	Позволяет управлять стилем выдачи текста в воздушном шарике. Некоторые биты этого свойства, будучи установлены в значение 1, означают включенность какого-либо параметра: Bit 0 — balloon-on (шарик отображается); Bit 1 — size-to-text (размер шарика подгоняется под размер текста); Bit 2 — auto-hide (шарик автоматически исчезает после того, как отработал); Bit 3 — auto-pace (текст в шарике выводится по словам); Bits 16—23 — управляют количеством символов в шарике (от 1 до 128, однако у автора задание, значения меньшего 8 символов, привело к возникновению ошибки); Bit 24—31 — управляет количеством строк в шарике (от 1 до 128)

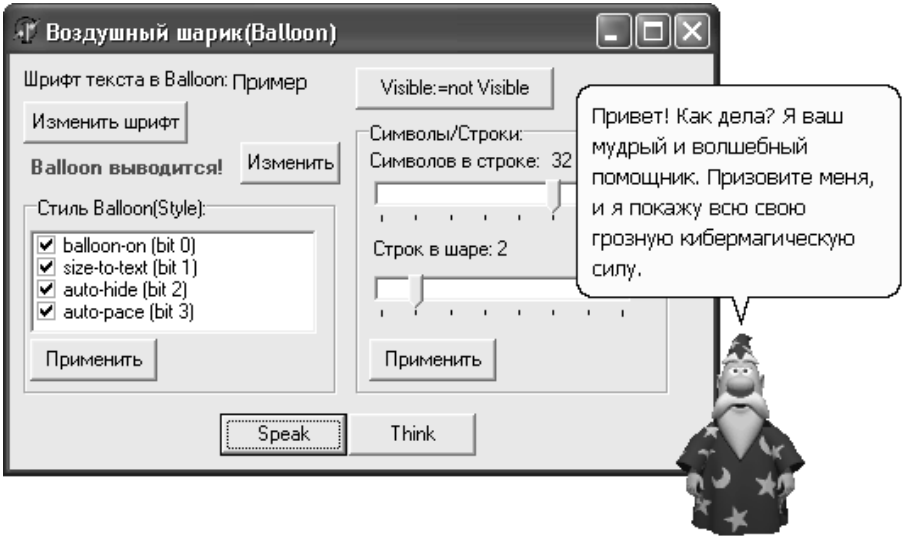


Рис. 2.39. Рабочее окно приложения, демонстрирующего свойства объекта `Balloon`

Рассмотрев код, приведенный в листинге 2.12, вы сможете разобраться более подробно с установкой и изменением параметров шаровоздушного средства Balloon.

Листинг 2.12. Модуль главной формы приложения, демонстрирующего управление параметрами "воздушного шарика" (проект каталога 27_Balloon)

```
unit Unit1;


interface


uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, ActiveX, OleCtrls, AgentObjects_TLB, ExtCtrls, StdCtrls,
CheckLst, ComCtrls;


type
  TForm1 = class(TForm)
    Agent1: TAgent;
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
    FontDialog1: TFontDialog;
    btSpeak: TButton;
    btThink: TButton;
    clbProp: TCheckListBox;
    Button2: TButton;
    trChars: TTrackBar;
    trString: TTrackBar;
    Button3: TButton;
    Button4: TButton;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Label3: TLabel;
    lbc: TLabel;
    Label5: TLabel;
    lbl: TLabel;
    lbOnOffBln: TLabel;
    Button5: TButton;
    procedure FormCreate(Sender: TObject);
```

```
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure btSpeakClick(Sender: TObject);
    procedure btThinkClick(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Agent1Click(Sender: TObject; const CharacterID: WideString;
      Button, Shift, x, y: Smallint);
    procedure Button4Click(Sender: TObject);
    procedure Agent1AgentPropertyChange(Sender: TObject);
    procedure Button5Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  Chars: IAgentCtlCharacterEx;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
  f, i: integer;
begin
  Agent1.Characters.Load('MyAgent', 'merlin.acs');
  Chars:=Agent1.Characters.Character('MyAgent') as IAgentCtlCharacterEx;
  Chars.Show(1);
  Label2.Font.Name:=(Chars.Balloon as IAgentCtlBalloonEx).FontName;
  Label2.Font.Size:=(Chars.Balloon as IAgentCtlBalloonEx).FontSize;
  f:=(Chars.Balloon as IAgentCtlBalloonEx).Style;
  for i:=0 to 3 do
   begin
    clbProp.Checked[i]:=(f mod 2)<>0;
```

```
    f:=f shr 1;
   end;
  f:=(Chars.Balloon as IAgentCtlBalloonEx).Style;
  f:=f shr 16;
  for i:=16 to 23 do
   begin
     if (f mod 2)<>0 then
      break
     else
      begin
       f:=f shr 1;
       trChars.Position:=trChars.Position+1;
      end;
   end;
  f:=(Chars.Balloon as IAgentCtlBalloonEx).Style;
  f:=f shr 24;
  for i:=24 to 31 do
   begin
     if (f mod 2)<>0 then
      break
     else
      begin
       f:=f shr 1;
       trString.Position:=trString.Position+1;
      end;
   end;
  lbc.Caption:=IntToStr(Chars.Balloon.CharsPerLine);
  lbl.Caption:=IntToStr(Chars.Balloon.NumberOfLines);
  if Chars.Balloon.Enabled then
   lbOnOffBln.Caption:='Balloon выводится!'
  else
   lbOnOffBln.Caption:='Balloon НЕ выводится!'
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Agent1.Characters.Unload('MyAgent');
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 FontDialog1.Font:=Label2.Font;
 if FontDialog1.Execute then
  begin
    Label2.Font:=FontDialog1.Font;
    Chars.Balloon.FontName:=Label2.Font.Name;
    Chars.Balloon.FontSize:=Label2.Font.Size;
  end;
end;

procedure TForm1.btSpeakClick(Sender: TObject);
begin
 Chars.Speak('Привет! Как дела? '+Chars.Description, '');
end;

procedure TForm1.btThinkClick(Sender: TObject);
begin
 Chars.Think('Привет! Как дела? '+Chars.Description);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
 f, i: Integer;
begin
 f:=(Chars.Balloon as IAgentCtlBalloonEx).Style;
 f:=f or 15; //Затираем единицами первые 4 бита
 f:=f xor 15; //Забиваем их нулями
 //Включаем необходимые биты с 0-го до 3-го:
 for i:=0 to 3 do
  if clbProp.State[i]=cbChecked then
   if i=0 then
    f:=f+1//Если первый бит 1, то просто прибавляем единицу
   else
    f:=f+(2 shl (i-1)); //иначе прибавляем степень 2
 (Chars.Balloon as IAgentCtlBalloonEx).Style:=f;
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
var
 f, i: Integer;
begin
 f:=(Chars.Balloon as IAgentCtlBalloonEx).Style;
 i:=$FFFFFFF0; //все биты равны 1, кроме битов с 0 по 3
 f:=f or i; //Затираем все биты единицами, кроме первых четырех
 f:=f xor i; //Обнуляем единичные биты, таким образом у нас остается
             //только информация о битах с 0-го по 3-й
 i:=(2 shl (trChars.Position-1)); //Включаем бит количества символов
 i:=i+(2 shl (trString.Position-1)); //Включаем бит для количества строк
 f:=f or i; //Прибавляем к ним сохраненные первые биты
 (Chars.Balloon as IAgentCtlBalloonEx).Style:=f;
 lbc.Caption:=IntToStr(Chars.Balloon.CharsPerLine);
 lbl.Caption:=IntToStr(Chars.Balloon.NumberOfLines);
end;

procedure TForm1.Agent1Click(Sender: TObject;
  const CharacterID: WideString; Button, Shift, x, y: Smallint);
begin
 Chars.Stop(EmptyParam);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
 Chars.Balloon.Visible:=not Chars.Balloon.Visible;
end;

procedure TForm1.Agent1AgentPropertyChange(Sender: TObject);
begin
 if Chars.Balloon.Enabled then
  lbOnOffBln.Caption:='Balloon выводится!'
 else
  lbOnOffBln.Caption:='Balloon НЕ выводится!'
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
```

```
 Agent1.PropertySheet.Visible:=true;
end;

end.
```

Напоследок дам совет: не забывайте проверять видимость воздушного шарика, если вы желаете сообщить пользователю нечто важное, а также поддерживаемость определенного шрифта (чтобы не получить каракули в тексте). Учитывая, что более функциональным "шаровоздушным" средством является приложение BalloonDialog от фирмы SommyTech®, не будем останавливаться подробно на стандартном объекте `Balloon`.

Синхронизация действий персонажей

До сего момента мы, в основном, имели дело с единственным персонажем в единственном же приложении и не касались принципов работы с несколькими агентами более чем в одном приложении. Настала пора поговорить и об этом.

Обычно в приложениях редко приходится использовать более одного персонажа, хотя такие ситуации вполне могут создаться. Расскажу о свойствах одной из них. Может быть, вы пожелаете создать "театр MS Agent", где действующими актерами станут персонажи Microsoft Agent. А что, это мысль! Актеры умеют говорить, могут обладать большим количеством анимаций, их довольно просто создавать и программировать.

Основываясь на наших знаниях, давайте попробуем подумать, как можно синхронизировать действия персонажей. Может быть, стоит при посылке запроса запоминать его идентификатор, затем отслеживать окончание запроса через событие `OnRequestComplete` и запускать следующий? Согласитесь, что схема не очень приятная и не самая очевидная. Поэтому ей предпочитают способ с применением специального метода объекта `Character` — `Wait`. Данный метод искусственно создает очередь анимаций для персонажа, чтобы тот ждал выполнения заданного специфического запроса на анимацию другого персонажа до тех пор, пока она не закончится. Иными словами, если требуется в одном коде запустить поочередно несколько запросов разных персонажей для создания эффекта последовательности действий, вы методом `Wait` заставляете один персонаж ждать окончания запроса другого, естественно, передавая в метод ссылку на объект `Request`. Например:

```
Merlin.Wait(Genie.Speak('Это мой друг Мерлин!', ''));
Merlin.Speak('Привет!', '');
```

В приведенном коде мы заставляем персонаж Merlin ждать выполнения запроса Genie, пока он не представит агента Merlin. По окончании выполне-

ния запроса `Speak` высказывается Merlin. Без применения метода `Wait` серия действий будет выполняться не последовательно, а одновременно.

В коде проекта каталога 28_Wait приведен пример более продолжительной сценки, в ходе которой персонаж Genie сначала представляется сам, затем представляет своего коллегу агента Merlin, который в свою очередь рассказывает немного о себе сам, пока Genie за ним наблюдает. По завершении речи Genie аплодирует, и Merlin исчезает. Посмотрите, как устроен код этой сцены:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   Agent1.Characters.Load('Genie', 'Genie.acs');
   Genie:=Agent1.Characters.Character('Genie') as IAgentCtlCharacterEx;
   Agent1.Characters.Load('Merlin', 'Merlin.acs');
   Merlin:=Agent1.Characters.Character('Merlin') as IAgentCtlCharacterEx;

   Genie.MoveTo(200, 200, 0);
   Genie.Show(0);

   Genie.Speak('Привет! '+Genie.Description, '');
   Genie.Play('GestureLeft');
   Merlin.Wait(Genie.Speak('А вот и мой друг Мерлин!', ''));
   //Мерлин ждет, пока его представит Джинн
   Merlin.MoveTo(400, 200, 0);
   Merlin.Show(0); //Теперь появляется сам Мерлин
   Genie.Play('RestPose');
   Merlin.Speak('Привет!', '');
   Genie.Play('LookLeft');
   Genie.Wait(Merlin.Speak('Спасибо за представление!', ''));
   Merlin.Wait(Genie.Play('Acknowledge'));
   Genie.Play('LookLeft');
   Genie.Wait(Merlin.Speak('Привет! '+Merlin.Description, ''));
   Merlin.Wait(Genie.Play('Congratulate'));
   Genie.Play('RestPose');
   Merlin.Speak('Пока!', '');
   Merlin.Play('Wave');
   Merlin.Hide(0);
end;
```

Рабочее окно приложения показано на рис. 2.40.

Рис. 2.40. Джинн ждет выполнения запроса `Speak` Мага

Надо заметить, что сам метод `Wait` также возвращает указатель на объект `Request`, поэтому его самого можно тоже прерывать и останавливать. В этом случае ожидание одного персонажа заканчивается, и он выполняет следующие действия в очереди (см. проект каталога 29_WaitStop):

```
req:=Merlin.Wait(Genie.Speak('А вот и мой друг Мерлин!', ''));
Merlin.Stop(req);
Merlin.Play('DoMagic1');
```

Последовательность действий данного примера будет нарушена вызовом `Stop`, поэтому визуально никакой разницы между этой сценой и последовательностью команд без метода `Wait` не будет.

> Метод `Interrupt` не прерывает выполнение метода `Wait`, т. е. персонаж продолжит ожидание. Происходит это по понятной причине — `Interrupt` ставится в очередь запросов для текущего персонажа. Для остановки ожидания следует пользоваться только методом `Stop`.

Если же требуется прервать выполнение запроса и перейти к следующим действиям сцены, то нужно останавливать не Wait-запрос, а запрос на анимацию:

```
req:=Genie.Speak('А вот и мой друг Мерлин!', '');
Merlin.Wait(req);
Merlin.Play('DoMagic1');
Genie.Stop(req);
```

Здесь речь персонажа Genie будет пропущена, и выполнение сцены продолжится со следующей команды (пример каталога 30_WaitPlayStop).

Много приложений + много персонажей

Перейдем к рассмотрению работы одного персонажа в нескольких клиентских приложениях, затем нескольких персонажей — в одном и, наконец, многих агентов — в нескольких приложениях.

Проведем эксперимент. Запустите тестовое приложение (рабочее окно показано на рис. 2.41) из каталога 31_ActiveClient\App1, пощелкайте на всех трех кнопках программы, посмотрите, какие результаты записываются в лог событий, не забудьте рассмотреть контекстное меню персонажа и его команды. Теперь запустите вторую копию приложения. Взгляните, каким образом изменилась работа функций. Вы сможете увидеть, как теперь в разных копиях приложения отображается информация о том, что переместило персонаж или какое приложение его скрыло. Все это выглядит логично, а вот попробуйте просто щелкнуть на ассистенте! Посмотрите — вторая копия сообщила о щелчке, а первая как будто и не почувствовала его. Теперь вместо второй копии этого приложения загрузите приложение из каталога 31_ActiveClient\App2. Обратите внимание: все странности повторяются, но теперь еще и контекстное меню персонажа изменилось, и его команды определяет только второе приложение. После закрытия второй копии первого приложения или второго приложения меню принимает свой прежний вид, и первая копия приложения, как и положено, начинает реагировать на щелчки мышью на агенте и командах меню.

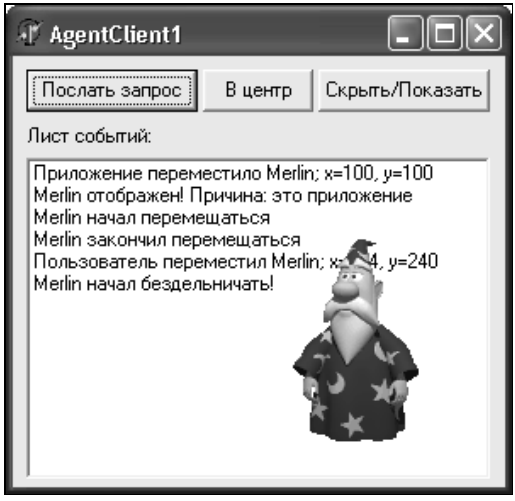


Рис. 2.41. Пример для эксперимента с активным клиент-приложением

Происходят подобные странности потому, что с поведением агента связано такое понятие, как *активный клиент* (Active client). Так как персонаж, имеющий уникальный GUID, может быть загружен единожды (ведь иначе пользователь не сможет отличить одну копию персонажа от другой), то следующее приложение, загружающее тот же самый персонаж, получает ссылку на уже загруженный и пользуется им как своим собственным. Активное клиентское приложение получает данные о большом числе событий, которых как мы видели, другие клиент-приложения не получают; изменения происходят с контекстным меню (что подразумевает и изменения в речевых командах).

Замечание

Активное приложение захватывает обработку всех событий мыши, таких как `OnClick`, `OnDblClick`, `OnDragStart`, `OnDragComplete`, а также событий речевых команд: `OnCommand`, `OnListenStart`, `OnListenComplete`.

Учитывая, что приложению, использующему анимированный персонаж, свойственно это новое (для нас) состояние (быть активным/неактивным клиентом по отношению к персонажу), необходимо научиться управлять им и определять его.

Каждое вновь запущенное приложение, использующее тот же самый персонаж, что и предыдущее, становится для персонажа активным клиент-приложением ("клиентом"). Если активное приложение выгружается, то статус active client приобретает приложение, запущенное непосредственно перед ним. Порядок следования очереди нельзя изменить, перемещая фокус между окнами приложений. Каким же пользовательским, а также программным способом возможно сделать это, мы узнаем позже.

Определить, имеет ли данный персонаж других клиентов, можно по логическому свойству `HasOtherClients` объекта `Character` (а какого же еще!), возвращающему значение `True`, если персонаж имеет других клиентов, использующих его, и `False` в противном случае. Свойство показывает наличие других клиентов как для приложения, являющегося активным (active client), так и для всех остальных. Посмотреть работу свойства `HasOtherClients` можно в паре аналогичных примеров из каталога 32_HasOtherClients.

Естественно, изменение текущего активного клиентского приложения персонажа происходит неожиданно, т. е. по воле пользователя или по команде со стороны других программ, поэтому, чтобы контролировать эту ситуацию, следует обрабатывать событие `OnActiveClientChange`.

```
type TAgentActiveClientChange = procedure(Sender: TObject; const Charac-
terID: WideString; Active: WordBool) of object;

property OnActiveClientChange: TAgentActiveClientChange;
```

В виде параметров здесь передается идентификатор персонажа и параметр `Active` интерфейса `IAgentCtlCharacterEx`, показывающий, как изменилось состояние клиента, т. е. является ли он активным (значение `True`) или стал неактивным для данного персонажа (значение `False`).

Без сомнения, в жизни возможны ситуации, когда выполнение кода в каком-либо месте программы должно зависеть от активности клиентского приложения. Для того чтобы лишний раз не отслеживать событие и не заводить под эти нужды отдельную переменную, советую сверять значения свойства `Active` объекта `Character`. Данное свойство имеет значение 0, если клиент не является активным по отношению к данному персонажу, значение 1 — при текущем активном клиенте. Однако свойство это не логического типа, оно может также принимать значение, равное 2, и через несколько абзацев мы узнаем, в каких именно ситуациях. Сейчас замечу лишь, что значение свойства `Active` является доступным только для чтения.

В ходе переключения между различными активными клиентами персонаж меняет состав своего контекстного меню. И будет достаточно неприятно, если станет невозможным обращение к командам другого приложения. По меньшей мере было бы странно, если бы этому в действительности нельзя было помешать. Возвратимся к началу данной главы *(см. разд. 2.1, подразд. "Компоненты пользовательского интерфейса")*, где отмечалось трехчастное строение контекстного меню (см. рис. 2.6). Первая часть — самая верхняя — стандартная, ниже первой разделительной черты помещены команды текущего активного приложения (с чем мы уже познакомились), а вот ниже второй, последней черты располагаются пункты переключения между командами других приложений, использующих данный персонаж. Пока мы этого в наших приложениях не замечали, т. к. не назначали подпись группе команд. Объект `Commands` текущего персонажа обладает свойством `Caption`, которому следует присвоить определенное значение и затем передать свойству `Visible` того же объекта значение, равное `True`.

```
Merlin.Commands.Caption:='Команды моего приложения';
Merlin.Commands.Visible:=true;
```

Теперь запустите последовательно приложения из каталога 33_AgentActivate. Если запустить только одно приложение (например, из подкаталога 33_AgentActivate\App1), ничего не изменится, но при загрузке второй его копии или второго приложения из каталога 33_AgentActivate\App2 к меню персонажа добавляются кнопки для переключения между остальными клиент-приложениями. По щелчку на этих пунктах текущие команды заменяются на команды выбранного приложения и обратно, при этом меняется активное приложение, о чем можно судить по записям в логе событий. Если у вас на компьютере есть приложение-переводчик Magic Gooddy, обязательно посмотрите как в нем организована система меню — его пункт также появляется в списке. На рис. 2.42 показаны рабочие окна трех приложений,

использующих персонаж Merlin: на фоне рабочего окна приложения Magic Gooddy — окна двух демо-приложений, первое с контекстным меню персонажа. Заметьте, в контекстном меню персонажа присутствуют два пункта для перехода между двумя другими клиентскими приложениями.

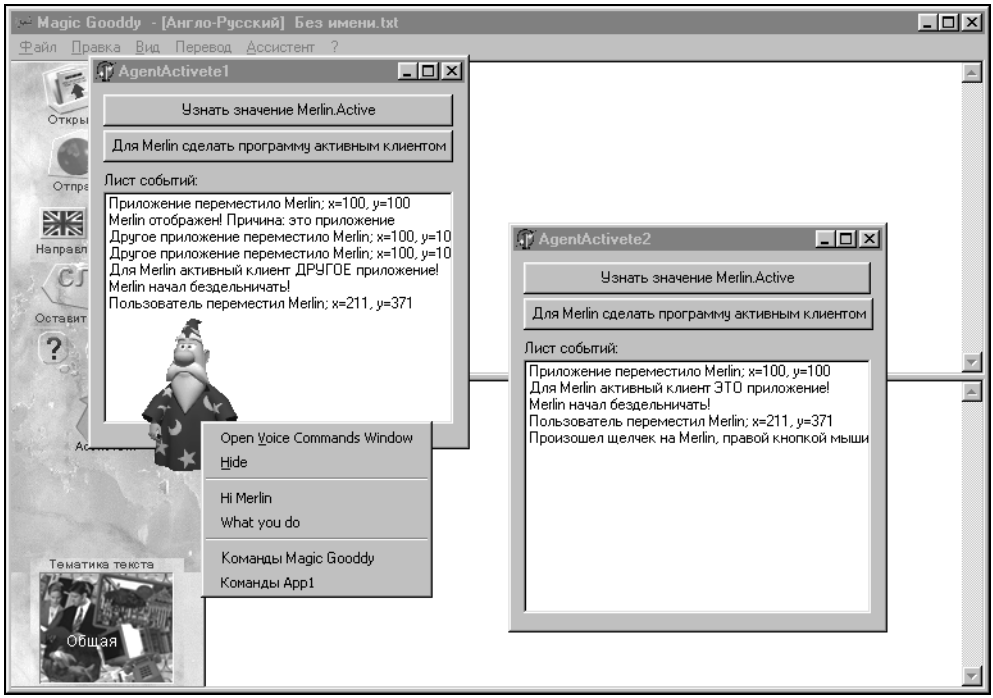


Рис. 2.42. Рабочие окна приложений, использующих персонаж Merlin

Важный совет

Предусматривайте в своем приложении переключение между активными клиентами, чтобы другое клиентское приложение не смогло захватывать персонаж на все время запуска. Обычно в случае использования команд в приложении должна быть установлена подпись группы команд меню; кроме того, нелишним будет вставлять в обработчик события активации формы `OnActivate` код изменения активного клиент-приложения, использующего персонаж, на свое приложение через вызов метода `Activate` с параметром, равным 1.

Назначение подписи и установку видимости пункта меню, ассоциированного с вашими командами, можно делать в любом месте программы, они вступают в силу, как только вы изменяете значения свойств. Даже если ваше приложение не является активным клиентом, в меню персонажа все равно появится нужный пункт.

Было бы неразумным со стороны Microsoft не включить поддержку изменения активного клиента программным способом; для этого как раз и существует специальный метод объекта `Character`:

```
function  Activate(State: OleVariant): WordBool; safecall;
```

Метод `Activate` изменяет активное клиент-приложение персонажа, и, очевидно, что в виде параметра может принимать значения, аналогичные значениям свойства `Active`: 0 — сделать клиент-приложение неактивным, 1 — сделать текущее клиент-приложение активным, и еще одно значение, так же как и в свойстве `Active` — 2, которое мы рассмотрим позже. Функция возвращает логическое значение, показывающее, удачно ли закончился метод.

Посмотрев на параметры этого метода и на их значения, кто-нибудь из читателей может задаться вопросом: "Если можно программно заставить приложение отказаться от роли активного клиента, то, следовательно, персонаж и вообще может не иметь активного клиентского приложения?" Нет, персонаж имеет всегда только один active client. Вызов метода `Activate` с параметром, равным нулю, будет означать, что вы пытаетесь передать активность другому приложению, т. е. тому, которое бы стало активным, если бы закрыли текущее приложение. Обращаю внимание на слово "пытаетесь", потому что в отсутствие альтернативного клиента метод будет завершен неудачей и узнать это можно будет по возвращаемому методом значению. Посмотреть реальную работу метода можно в приложениях проекта каталога 33_AgentActivate, а также (со значением 0) в проекте каталога 34_Activate0.

В этом подразделе я откладывал описание значения параметра `Active`, равного 2, а также значения параметра `State` в методе `Activate` именно для того, чтобы перейти к их рассмотрению в наиболее удобном месте. Так вот, сейчас мы к ним и перейдем. В некоторых примерах данной главы уже использовалось одновременно несколько персонажей в одном приложении, и если читатели внимательно изучали код, то наверняка заметили, что для удобства обращения к конкретному агенту (одному из нескольких) мы применяли несколько ссылок на интерфейсы персонажей `IAgentCtlCharacterEx`, например:

```
var
  Merlin: IAgentCtlCharacterEx;
  Genie: IAgentCtlCharacterEx;
begin
  Agent1.Characters.Load('Merlin', 'merlin.acs');
  Agent1.Characters.Load('Genie', 'genie.acs');
  Merlin:=Agent1.Characters.Character('Merlin') as IAgentCtlCharacterEx;
  Genie:=Agent1.Characters.Character('Genie') as IAgentCtlCharacterEx;
  Merlin.MoveTo(100, 100, 0);
```

```
  Genie.MoveTo(300, 100, 0);
  Merlin.Show(0);
  Genie.Show(0);
end;
```

В данном примере загружаются два персонажа, и оба отображаются друг с другом рядом. Давайте подумаем о функционировании нескольких персонажей в рамках одного приложения. Естественно, что для всех персонажей, используемых программой, она будет являться их активным клиентом (пока мы рассматриваем только одно приложение). Проблем с обработкой событий не должно быть, т. к. мы видели, что во всех событиях, посылаемых сервером MS Agent для конкретного персонажа, передается его идентификатор, т. е. имя, которое мы присвоили ему при загрузке. Недоразумений с вызовом методов конкретного персонажа тоже не будет, ведь к указателю на интерфейс каждого возможно обратиться через его имя:

```
AgentControl.Characters.Character('Merlin').метод_или_свойство
```

Стоит поинтересоваться, какой же персонаж из нескольких используемых будет, к примеру, принимать речевой ввод по нажатию горячей клавиши? Думаю, что и те читатели, которые совершенно не задумались над ответом, и те, кто пересмотрел все примеры перед принятием решения, могут ответить совершенно правильно. Ясно, что в данном случае отреагирует только тот персонаж, который был выбран последним, попросту щелчком мыши. Действительно, наши привычки, выработанные в процессе работы в графических операционных системах с оконным интерфейсом, уже залегли глубоко внутри, и опыт наблюдения за окнами приложений подсказывает, что обычно последнее выбранное приложение, т. е. активное, получает от пользователя ввод клавиатуры и мыши. И персонажи MS Agent, являясь по своей природе теми же окнами (неправильной формы), так же ведут себя в системе, правда, только по отношению к приложениям, использующим их, поскольку все они располагаются поверх остальных окон. Хотя и видим мы их как будто на одном уровне, все загружаемые персонажи выстраиваются по мнимой оси z трехмерной системы координат, перпендикулярной плоскости экрана. Говорят, что самый верхний персонаж, как бы самый ближний к нам, находится в состоянии, которое называется в англоязычной документации *input-active*, что буквально переводится как "активный для ввода".

Персонаж, активный для ввода, обладает некоторым приоритетом — он принимает все речевые команды пользователя, т. е. при нажатии горячей клавиши активизируется функция распознавания речи именно активного персонажа, и распознаются только его команды и, естественно, глобальные. При использовании нескольких персонажей одновременно в программе только один из них может быть активен для ввода, т. е. находиться в состоянии input-active.

Замечание

О распознавании речи средствами MS Agent мы поговорим в следующей главе, однако я намеренно пишу о тонкостях в работе с активным персонажем с учетом распознавания речи, чтобы не разбрасывать информацию по разным разделам, а также для того, чтобы читатель не забыл об этих нюансах и обратил на них внимание в будущем.

Эти уровни естественно мыслятся только у нас в голове, т. е. не существует каких-либо дополнительных свойств, сообщающих о том, на каком уровне располагается персонаж по отношению к другим. Самыми высокими приоритетами обладает только input-active персонаж, все остальные равны по значимости между собой (конечно, до момента исчезновения текущего активного персонажа). Однако интересно посмотреть, как агенты располагаются один под другим наподобие окон, поэтому обратите внимание на рис. 2.43.

Рис. 2.43. Скриншот окна, показывающий относительное расположение трех агентов один поверх другого

Давайте разберемся, как агент приобретает свойство активного для ввода персонажа и как может потерять его. Мы уже поняли, что изменить активный персонаж может пользователь, выбрав его мышью. Кроме того, каждый агент, отображаемый методом `Show` (сюда входит и отображение с помощью меню), становится input-active персонажем, причем не играет роли, появляется ли он впервые или после скрытия. Соответственно, потерять свойство активности агент может при скрытии методом `Hide` (а также через контекстное меню и голосовую команду), в этом случае input-active персонажем становится агент, стоящий на уровень ниже, т. е. тот, который до этого был активным. Сменить активного персонажа можно и голосовыми командами, если они доступны, и, кроме того, если доступен речевой ввод — в этом случае достаточно нужного агента позвать голосом по его имени, например,

Мага позвать по его английскому имени Merlin. При удачном завершении распознавания персонаж отобразится на экране и станет активным для принятия последующих команд.

Ранее, говоря об активном клиентском приложении, мы уже отмечали, что для каждого персонажа существует свое активное клиент-приложение, причем для нескольких персонажей активным клиентом может быть одно и то же приложение. Не нужно путать активное приложение с активным для ввода персонажем. Активное приложение обрабатывает все события пользовательского ввода, а получает события клавиатуры, мыши и голоса именно input-active персонажа, и именно его идентификатор передается в обработчики событий в виде параметра.

Теперь читатель сам может понять, о чем говорит значение свойства `Active` и параметра `State` в методе `Activate`, равное 2. Если свойство `Active` возвращает значение 2, то это значит, что текущий агент является самым верхним, т. е. input-active персонажем. Передавая с методом `Activate` параметр `State`, равный 2, мы пытаемся программным способом заставить определенного агента стать активным для ввода. Конечно, надо помнить, что в этом случае выполнение метода может завершиться неудачей, если персонаж был скрыт.

Так же, как и при смене активного клиента, при замене активного персонажа клиенту, чей персонаж стал активным, посылается событие `OnActivateInput`, а клиенту, персонаж которого только что избавился от этого состояния (перешел в состояние non-input-active), посылается событие `OnDeactivateInput`, конечно же, в виде параметров этих событий.

Для того чтобы "поиграть" с некоторыми свойствами и событиями, можно использовать демонстрационное приложение из каталога 35_InputActive (рабочее окно показано на рис. 2.44). При запуске демо-приложения сначала появляется сообщение о том, что Merlin (Маг) активен, затем, что тот же персонаж потерял активность ввода и сразу же за ним информация о том, как Genie (Джинн) стал input-active. Это доказывает то, что каждый персонаж, отображаемый методом `Show`, становиться активным для ввода. Пощелкайте мышью на персонажах и посмотрите, как передается активность ввода от одного к другому после щелчка.

Откройте окно команд голосом и обратите внимание, как меняется список команд при переходе от одного агента к другому. Этот факт доказывает, что выбранный персонаж действительно стал активным. Выберем персонаж Merlin, нажмем горячую клавишу (обычно <Scroll Lock>, если вы ее не меняли) для распознавания речи и произнесем команду "Hello, Merlin". В списке событий отобразится соответствующая информация, если команда была успешно распознана.

Рис. 2.44. Рабочее окно приложения, демонстрирующего свойства input-active персонажа в одном приложении

Замечание

Не забудьте, что для распознавания речи необходимо иметь установленный пакет функций Speech API, а также голосовой движок распознавания речи. Кроме того, к звуковой плате компьютера должен быть подключен микрофон. В ходе проверки примеров не забудьте включить его физически и программно через **Регулятор громкости**, открываемый двойным щелчком на значке динамика в системном лотке

В описываемом приложении персонаж Merlin имеет две команды — "Hello, Merlin" и "What are you", персонаж Genie тоже две — "Hello, Genie" и "Speak time". Попробуйте сказать Мерлину: "Speak time", — команда не будет распознана, потому как ее нет у активного персонажа. Однако если вы позовете Джинна, т. е. подадите речевую команду "Genie", и Джинн станет активным, о чем будут свидетельствовать записи в листе событий, то команда "Speak time" должна распознаваться успешно.

Реализацию программного кода предлагаю рассмотреть читателям самостоятельно, т. к. техническая его сторона несложна.

Замечание

Я намеренно не описываю подробности реализации программного кода приложений с поддержкой распознавания речевых команд, т. к. это тема следующего раздела, сейчас же для нас главное — уяснить влияние признака input-active на поведение персонажа.

Рассмотрим случай с несколькими приложениями и несколькими персонажами, тем самым достигнув финала данного раздела. Ничего особенного при использовании большого количества персонажей в нескольких приложениях одновременно не происходит. Дело в том, что следует сделать всего одно замечание по поводу input-active персонажей — данный признак глобален для любого персонажа, независимо от того, какое клиент-приложение его использует, однако именно активный клиент данного персонажа, являющегося input-active, обрабатывает все события относительно речевого ввода. То есть при условии взаимодействия двух приложений, один из агентов которых — общий, в том случае, если совместно используемый персонаж является input-active, голосовые команды будет воспринимать только приложение, служащее для него активным клиентом. Из второго же приложения свойство `Active` будет показывать, что оно не является активным. С другой стороны, если из определенного приложения значение свойства `Active` персонажа показывает, что он является самым верхним, то это означает, что данное приложение является для него также и активным. Это логично, поэтому не существует отдельного свойства для определения активности персонажа. Раз пользователь выбрал персонаж, передал ему этот своеобразный фокус ввода, значит, какое-то приложение является для него активным клиентом, и значит, по отношению к нему персонаж будет input-active.

Замечание

Единственный ассистент в системе всегда является input-active персонажем.

Для демонстрации использования многих персонажей создан пример, находящийся в каталоге 36_MoreAgent (рис. 2.45). Здесь можно загружать любое количество персонажей, выгружать их, программным путем изменять активное приложение каждого персонажа, а также задавать input-active персонаж.

Подытожим некоторые наши наблюдения. К персонажам MS Agent применимо такое понятие, как input-active. Приложение обрабатывает речевые команды только активного персонажа (активного по отношению к неактивным, если используется несколько агентов).

В случае работы нескольких приложений с одним и тем же персонажем одно из них (последнее, загрузившее персонаж) будет являться активным клиентом для персонажа, т. е. оно будет получать все события, посылаемые сервером. Это означает, что некоторые события не будут посланы остальным клиентским приложениям (события `OnClick`, `OnDblClick`, `OnDragStart`, `OnDragComplete`, `OnDragStart`); в других же событиях по значениям отдельных параметров можно установить, выполнено ли действие из другого приложения (события `OnShow`, `OnHide`, `OnMove`, `OnHelpComplete`, `OnListenStart`, `OnListenComplete`, `OnActiveClientChange`). Однако некоторые оставшиеся неперечисленными события посылаются только определенным клиентам —

такие, например, как события `OnRequestStart` и `OnRequestComplete`, предназначенные только для приложений, запустивших запрос.

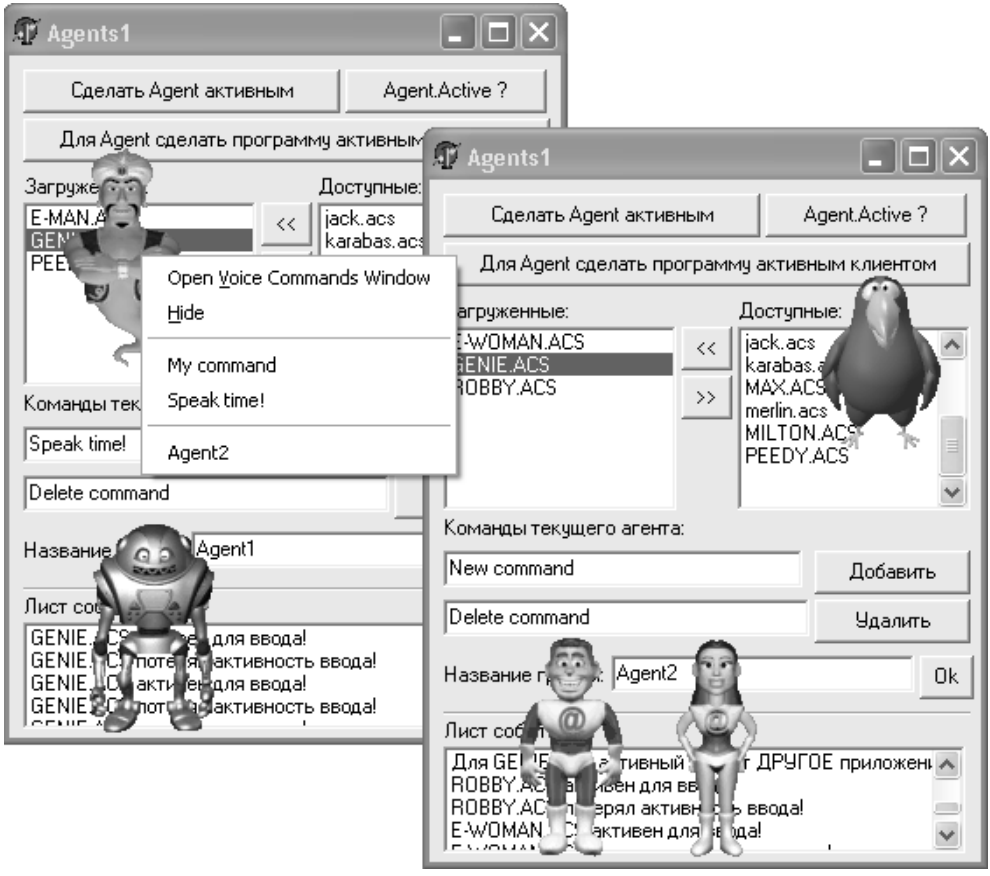


Рис. 2.45. Рабочее окно приложения, демонстрирующего работу персонажей одновременно в нескольких приложениях

Менять активный — input-active — персонаж можно, просто выбирая его мышью или вызывая его по имени голосом, а также программно — через метод `Activate` с параметром, равным 2. То же самое верно и для активного клиентского приложения какого-либо агента: автоматически активным становится приложение, последним загрузившее персонаж. При помощи контекстного меню персонажа можно переключаться между командами соответствующего приложения, и, следовательно, изменять активное клиент-приложение. В программном способе также используется метод `Activate` требуемого персонажа.

2.8. Речевой вывод и ввод

Тема данного раздела наиболее интересна своей необычностью, т. к. не существует практически ни одного руководства по столь новой технологии. Однако подробнее всего мы будем рассматривать ее в следующей главе, а пока научимся вводить речевые команды и пользоваться синтезом речи с помощью средств, предоставляемых MS Agent. Естественно, для более полного понимания идеи речевых технологий следует обратиться к *гл. 3*.

Название данного раздела говорит о том, что он посвящен описанию речевых возможностей технологии анимированного персонажа, однако в нем мы также узнаем, как включать и выключать спецэффекты агента, и познакомимся с такими новыми для нас объектами модели MS Agent, как `AudioOutput`, `SpeechInput`, `CommandsWindow`, являющимися дочерними объектами элемента управления Agent Control.

В начале этой главы мы говорили о том, что не следует злоупотреблять речевым вводом и выводом, т. к. технология распознавания речи не очень распространена, да и компоненты ее "не из легких" (по занимаемому объему памяти). В Windows 2000/XP к услугам англоязычных пользователей имеется все необходимое для синтеза речи, нам же либо придется всерьез изучать английский, либо отказываться от данной технологии. С синтезом речи дело тоже обстоит не совсем гладко, однако синтез русской речи возможен и дает неплохие результаты. Вы можете спросить, зачем тогда обо всем этом писать, если и технология не распространена, и компоненты "тяжелые", и качество распознавания невысокое. На это есть несколько причин. Во-первых, некоторым разработчикам мультимедийных и развлекательных приложений (в частности, игр) хотелось бы использовать подобные возможности. Во-вторых, Microsoft не стоит на месте и новые речевые пакеты будут появляться, да мы не вечно же будем "сидеть под Windows 98". Ну, и наконец, последняя причина — отсутствие документации на русском языке, как переводов, так и простых руководств. В Интернете, конечно, имеются кое-какие статьи, которые могут отчасти помочь в освоении этих технологий (в том числе и MS Agent), однако там говорится далеко не обо всех нюансах, вследствие чего конечному разработчику приходится проводить множество экспериментальных проверок.

Базовые понятия

Я уже упоминал о программных компонентах, которые присутствуют не во всяких операционных системах. В их число входят пакет речевых функций Speech API и речевые движки. В следующей главе мы остановимся на них подробнее, сейчас же коротко поясним суть самих компонентов и опишем порядок их установки.

Уверен, все читатели понимают, что многие дополнительные функции предоставляются специальными компонентами. Как известно, одной из наиболее интересных технологий, служащих для установления взаимодействия между различными приложениями, является COM-технология и ее дочерние технологии. Так и для синтеза речи в системе должны присутствовать некоторые COM-объекты, которые реализуют речевые функции. Именно поэтому пакет, устанавливающий набор необходимых компонентов, называют *пакетом речевых функций.* Это Speech API. Ссылка для загрузки его из Интернета приведена в *приложении 1.* Для установки пакета следует запустить установочный файл, после чего без всяких проблем должна пройти инсталляция.

Сразу после установки пакета Speech API еще нельзя использовать речевые функции в своих приложениях, вернее их можно использовать в смысле получения доступа к ним, но программа будет неработоспособной. Как и в случае с MS Agent, когда помимо самого сервиса нам нужны были конкретные персонажи, так и здесь, кроме компонента Speech API, необходимы речевые синтезаторы и специальные модули распознавания речи. Так же как при выборе персонажа существует возможность выбирать его из набора имеющихся, так и речевыми функциями разрешается смена речевого синтезатора на тот, который наиболее качественно воспроизводит речь и больше нас устраивает. Существует много модулей распознавания речи и модулей, по-разному воспроизводящих речь, и благодаря этому многообразию мы можем подобрать более качественный модуль и использовать именно его. Подобный компонентный подход удобен: функции не исчезают, меняются лишь используемые модули. Это обеспечивает мобильность, т. е. программным путем на компьютере конечного пользователя впоследствии возможно будет определить установленные синтезаторы и использовать именно их.

Речевые синтезаторы и модули распознавания речи иначе называют *engine*, что с английского переводится как "двигатель", поэтому в Интернете они часто называются "движками". При этом слово "движок" дополняется указанием на его тип — "движок синтеза речи" или "движок распознавания речи". Все движки различаются по фирмам-изготовителям, техническим характеристикам, а также по качеству выполнения синтеза и распознавания речи.

Объект *AudioOutput*

Объект `AudioOutput` элемента управления Agent Control обеспечивает связь с аудиосвойствами, реализуемыми сервером. Соответствующее свойство элемента управления возвращает указатель на интерфейс `IAgentCtlAudioObjectEx`, обеспечивающий связь с этим объектом:

`AgentControl``.AudioOutput.`*`свойства`*

Это расширенный интерфейс, хотя элемент управления Agent Control поддерживает еще и старый интерфейс `IAgentCtlAudioObject` В новом интерфейсе добавлено одно свойство, позволяющее узнать состояние аудиоканала. Сразу отмечу, что все три свойства данного объекта предназначены только для чтения: значения двух из них может изменять пользователь через диалоговое окно **Дополнительные параметры персонажей** (Advanced Character Options) и значение свойства состояния аудиоканала самостоятельно изменяется при определенных состояниях. Значения всех свойств представлены в табл. 2.15.

Таблица 2.15. *Свойства объекта* `AudioOutputt`

Свойство	Описание
`property Enabled: WordBool`	Показывает, включен ли речевой вывод. Отражает состояние флажка **Воспроизводить речь** (Play Audio Output) на вкладке **Вывод** (Ouput) диалогового окна **Дополнительные параметры персонажей** (Advanced Character Options)
`property SoundEffects: WordBool`	Свойство отражает состояние вывода звуковых эффектов, волновых звуков, заданных при воспроизведении анимации. Свойство показывает состояние флажка **Воспроизводить звуковые эффекты персонажа** (Play Character Sound Effects) на вкладке **Вывод** (Output) диалогового окна **Дополнительные параметры персонажей** (Advanced Character Options)
`property Status: Smallint`	Возвращает состояние аудиоканала: 0 — аудиоканал доступен (не занят); 1 — аудиовывод не поддерживается (например, не установлена звуковая карта); 2 — невозможно открыть аудиоканал, т. к. он занят (возможно, другим приложением); 3 — аудиовывод занят потому, что сервер обрабатывает речевой ввод пользователя; 4 — аудиоввод занят, т. к. персонаж в текущий момент говорит; 5 — аудиоканал занят, т. к. он ждет пользовательского ввода речи; 6 — обнаружена другая неизвестная проблема, которая помешала обратиться к аудиоканалу

Как видно из таблицы, первые два свойства отражают состояние опций в диалоговом окне **Дополнительные параметры персонажей** (Advanced Character Options), а значение последнего свойства изменяется в процессе работы.

Поэтому в демонстрационном приложении подкаталога 37_AudioOutput отслеживание изменений значений первых двух свойств реализовано через обработку события `OnAgentPropertyChange`, а состояние последнего считывается через каждый такт таймера `Timer1`.

Попробуйте при испытании приложения (его рабочее окно приведено на рис. 2.46) включать и отключать звуковой вывод и посмотрите, как это сказывается на чтении персонажем текста. При отключенном свойстве `Enabled` персонаж перестает читать, рядом только появляется воздушный шарик, а агент даже не шевелит губами (и в самом деле — зачем напрягаться?).

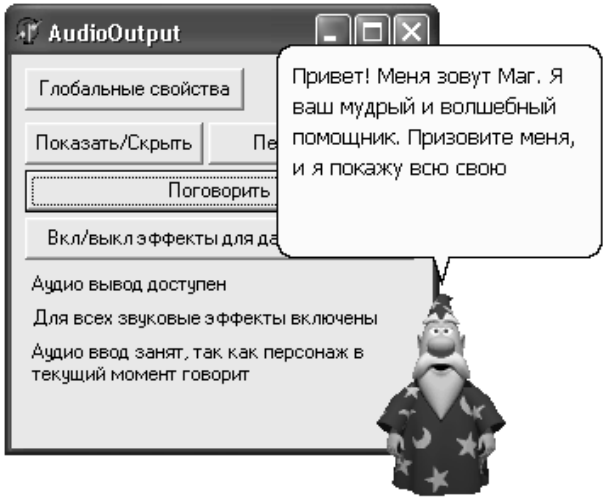


Рис. 2.46. Рабочее окно приложения, демонстрирующего свойства объекта `AudioOutput`

Читатели, наверное, уже не раз замечали, что некоторые анимации сопровождаются звуковыми эффектами. Обычно это анимации Showing, Hiding, Moving и др. Волновые файлы (файлы формата WAV), содержащие звук, сопровождающий анимацию, указываются при создании персонажа. По умолчанию все звуковые эффекты воспроизводятся во время анимации, однако этой опцией можно управлять и с помощью флажка **Воспроизводить звуковые эффекты персонажа** (Play Character Sound Effects) диалогового окна **Дополнительные параметры персонажей** (Advanced Character Options) (вкладка **Вывод**), а программно узнать, включены или отключены эффекты, можно через свойство `SoundEffects`.

Наибольшее практическое значение для нас имеет, конечно, свойство `Status`, отражающее состояние аудиоканала. Если в программе требуется выводить речь, то нелишним будет проверить, сможет ли пользователь нас

услышать? Если у пользователя нет звуковой платы (что маловероятно, конечно), то зачем напрягаться и тратить время на запуск и выполнение вывода речи методом `Speak`, лучше уж сразу выдавать соответствующие сообщения. Да мало ли примеров можно привести.

Если необходимо отключить или включить воспроизведение звуковых эффектов только для определенного персонажа, то следует пользоваться свойством `SoundEffectsOn` объекта `Character`:

```
Chars.SoundEffectsOn:=true;
```

Данное свойство доступно также и для записи. Пользуясь им, следует помнить, что изменение состояния данного свойства влияет на все клиентские приложения, использующие тот же персонаж.

Учимся говорить

В предыдущих примерах мы несколько раз сталкивались с методом `Speak`, заставляющим персонаж читать любой текст. Настал момент официально познакомиться с этим методом:

```
function  Speak(Text: OleVariant; Url: OleVariant): IAgentCtlRequest;
```

С первым параметром передается текст для чтения, с вторым — волновой файл, который следует воспроизводить при отображении необходимых слов, если синтез речи не требуется. Это может быть обычный волновой файл формата WAV или лингвистический расширенный звуковой файл (linguistically enhanced sound file, LWV), который создается в специальном лингвистическом редакторе *(см. гл. 4)*. Метод также возвращает указатель на объект `Request`, т. е. метод можно прерывать, останавливать, удалять из очереди и т. п.

Следующий код заставляет персонаж прочитать текст из текстового поля редактирования:

```
Chars.Speak(Edit1.Text, '');
```

Интересных эффектов можно добиться с помощью звуковых файлов, и в предыдущем разделе это демонстрировалось на примере раздельного формата персонажа, когда мы через HTTP-сервер загружали звуковые файлы, а затем их воспроизводили. Так вот, подобным приемом можно пользоваться и работая с обычным персонажем (структурным файлом ACS). В этом случае, естественно, не следует предварительно загружать звуковой файл, нужно просто вызвать метод `Speak` с полным именем звукового файла. Например:

```
Chars.Speak('Расскажи, Снегурочка, где была? Расскажи-ка, милая, как дела?',
   'c:\Sound1.wav');
```

Посмотрите, как забавно реализовано пение в примере каталога 38_Speak (рабочее окно приложения показано на рис. 2.47).

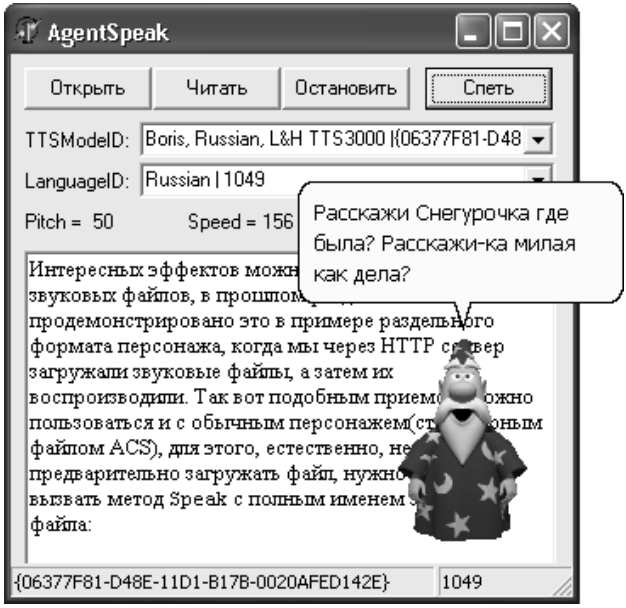


Рис. 2.47. Рабочее окно приложения, демонстрирующего речевые функции

Интересна особенность метода `Speak`. Нередко бывают ситуации, когда не очень интересно заставлять агента произносить в определенные моменты времени одни и те же слова. Ранее мы отмечали, что для большего "очеловечивания" персонажа следует использовать синонимический ряд; так вот, вместо того чтобы придумывать и реализовывать свои собственные алгоритмы выбора случайных фраз, обратим внимание на то, что метод `Speak` уже обладает такой возможностью. Ничего особенного делать не нужно, просто следует передавать методу весь синонимический ряд в виде текста, разделяя каждую фразу вертикальной чертой "|", например, так:

```
Chars.Speak('Привет!|Hello!|Здравствуйте!|Hi!|Как дела?', '');
```

В случае синтеза речи на основе переданного текста MS Agent скрытно от нас, используя функции Speech API и возможности голосовых синтезаторов, преобразует его в речь. Основные параметры в персонаж закладываются при его создании — GUID речевого синтезатора по умолчанию, тембр голоса и скорость речи. Естественно, что параметры эти можно изменять, и постепенно мы узнаем о том, как это делается.

GUID текущего используемого речевого синтезатора персонажа хранит свойство `TTSModeID` объекта `Character` (`IAgentCtlCharacterEx`):

```
property TTSModeID: WideString
```

Для того чтобы сменить текущий используемый речевой двигатель, всего лишь нужно изменить данное свойство, например, так:

```
Merlin.TTSModeID:= '{06377F81-D48E-11d1-B17B-0020AFED142E}';
```

В данном примере текущий голосовой движок изменяется на синтезатор русской речи от фирмы Lernout&Hauspie — Boris, Russian, L&H TTS3000[7]. В следующей таблице (табл. 2.16) собраны идентификаторы наиболее популярных движков, однако таким способом их выбора вряд ли кто пользуется, поэтому в следующей главе мы рассмотрим способы того, как найти все установленные в системе речевые синтезаторы и определить их GUID.

Таблица 2.16. *Идентификаторы голосовых движков*

Голос	TTSModeID
Adult Female #1, US English, L&H TruVoice	{CA141FD0-AC7F-11D1-97A3-006008273008}
Adult Female #2, US English, L&H TruVoice	{CA141FD0-AC7F-11D1-97A3-006008273009}
Adult Male #1, US English, L&H TruVoice	{CA141FD0-AC7F-11D1-97A3-006008273000}
Adult Male #2, US English, L&H TruVoice	{CA141FD0-AC7F-11D1-97A3-006008273001}
Adult Male #3, US English, L&H TruVoice	{CA141FD0-AC7F-11D1-97A3-006008273002}
Adult Male #4, US English, L&H TruVoice	{CA141FD0-AC7F-11D1-97A3-006008273003}
Adult Male #5, US English, L&H TruVoice	{CA141FD0-AC7F-11D1-97A3-006008273004}
Adult Male #6, US English, L&H TruVoice	{CA141FD0-AC7F-11D1-97A3-006008273005}
Adult Male #7, US English, L&H TruVoice	{CA141FD0-AC7F-11D1-97A3-006008273006}
Adult Male #8, US English, L&H TruVoice	{CA141FD0-AC7F-11D1-97A3-006008273007}
Carol, British English, L&H TTS3000	{227A0E40-A92A-11d1-B17B-0020AFED142E}

[7] Данное составное название представляет собой краткое описание движка с его основными параметрами. Boris — имя чтеца, Russian — язык синтеза, L&H — краткое наименование фирмы, TTS3000 — модель движка.

Таблица 2.16 *(окончание)*

Голос	TTSModeID
Peter, British English, L&H TTS3000	{227A0E41-A92A-11d1-B17B-0020AFED142E}
Svetlana, Russian, L&H TTS3000	{06377F80-D48E-11d1-B17B-0020AFED142E}
Boris, Russian, L&H TTS3000	{06377F81-D48E-11d1-B17B-0020AFED142E}
Anna, German, L&H TTS3000	{3A1FB760-A92B-11d1-B17B-0020AFED142E}
Stefan, German, L&H TTS3000	{3A1FB761-A92B-11d1-B17B-0020AFED142E}
Veronique, French, L&H TTS3000	{0879A4E0-A92C-11d1-B17B-0020AFED142E}
Pierre, French, L&H TTS3000	{0879A4E1-A92C-11d1-B17B-0020AFED142E}

Изменение свойства `TTSModeID` вызывает эффект только в вашем клиентском приложении, т. е. не влияет на другие программы, и тем более на персонажи.

Не менее важными характеристиками речи являются тембр голоса и скорость произнесения слов, однако те свойства, которыми обладает расширенный интерфейс объекта `Character`, доступны только для чтения.

Свойство `Pitch`, возвращающее значение тембра голоса, передает целочисленное значение тона речи в герцах (Гц):

```
property Pitch: Integer
```

Свойство скорости речи `Speed` определяет количество слов в минуту (WPM, word per minute):

```
property Speed: Integer
```

Еще раз повторяю, что оба эти свойства (так же как и GUID синтезатора по умолчанию) закладываются в ходе создания персонажа. Однако если речевой синтезатор можно менять, просто присваивая свойству другой идентификатор класса движка, то с тембром голоса и скоростью речи дело обстоит сложнее. Принципы изменения параметров чтения мы узнаем в *подразд. "Коррекция речи" данного раздела*, а пока я расскажу еще об одном свойстве, которое также нам пригодится, — о свойстве `LanguageID`, определяющем идентификатор языка:

```
property LanguageID: Integer
```

Данное свойство управляет языковым признаком персонажа, влияет на выдаваемые диалоги, вызов ошибки, сообщения, подписи, язык текстов воздушного шарика, названия в контекстном меню стандартных пунктов, на язык распознавания речи и язык синтеза речи. Все это важно учитывать. Свойство имеет значение, соответствующее текущему языку в системе, если установлен национальный модуль, иначе его значение соответствует английскому языку — 1033 (English (US)).

Некоторые значения данного свойства приведены в табл. 2.17.

Таблица 2.17. *Языковые контанты*

Язык	Hex-значение	Десятичное значение
English (British)	$0809	2057
English (US)	$0409	1033
French	$040C	1036
German	$0407	1031
Italian	$0410	1040
Russian	$0419	1049
Chinese (упрощенный)	$0804	2052
Chinese (традиционный)	$0404	1028
Japanese	$0411	1041

При неудачной смене языка, т. е. если национальный языковый модуль не установлен или если отсутствует шрифт для поддержки отображения информации, сервер генерирует исключение и восстанавливает последнее значение свойства `LanguageID`.

Замечание

Обратите внимание — ошибка смены языка не вызывается, если отсутствует соответствующий движок распознавания или синтеза речи.

Напоследок предлагаю поиграть с примером каталога 38_Speak, в котором реализованы почти все рассмотренные свойства. Советую не загружать в метод `Speak` слишком большие тексты, порядка нескольких десятков килобайт, т. к. загружаться такой текст будет очень долго. Для более "продвинутого" чтения лучше всего использовать разбиение текста, например, на абзацы или предложения, и по завершении чтения одного куска загружать следующий. Окончание чтения одного фрагмента можно отслеживать по завершению выполнения запроса, например, при запуске метода `Speak` запоминать идентификатор ID запроса и по завершении по-

следнего загружать новый фрагмент. Такой способ чтения как раз и реализован в предлагаемом примере. Рассмотрим фрагмент кода, приведенный в листинге 2.13.

Листинг 2.13. Фрагмент кода примера каталога 38_Speak

```
unit Unit1;

interface
...
type
  TForm1 = class(TForm)
    Agent1: TAgent;
    btRead: TButton;
    btStop: TButton;
    RichEdit1: TRichEdit;
  ...
  end;
...
var
  Form1: TForm1;
  Chars: IAgentCtlCharacterEx;
  Req: IAgentCtlRequest;
  t: String;
  RID: integer;
  Reading: Boolean;
...
implementation

procedure TForm1.btReadClick(Sender: TObject);
var
 s: string;
begin
 Reading:=true;
 t:=RichEdit1.Text;
 if pos('.', t)<>0 then
  s:=copy(t, 1, pos('.', t))
 else
  s:=t;
```

```
 Req:=Chars.Speak(s, '');
 RID:=Req.ID;
 if pos('.', t)<>0 then
  delete(t, 1, pos('.', t))
 else
  t:='';
end;

procedure TForm1.btStopClick(Sender: TObject);
begin
 Reading:=false;
 Chars.Stop(Req);
end;

procedure TForm1.Agent1RequestComplete(Sender: TObject;
  const Request: IDispatch);
var
 R: IAgentCtlRequest;
 s: string;
begin
 Request.QueryInterface(IID_IAgentCtlRequest, R);
 if(R<>nil) and(R.ID=RID) and(Reading) and(t<>'') then
  begin
    if pos('.', t)<>0 then
     s:=copy(t, 1, pos('.', t))
    else
     s:=t;
    Req:=Chars.Speak(s, '');
    RID:=Req.ID;
    if pos('.', t)<>0 then
     delete(t, 1, pos('.', t))
    else
     t:='';
  end;
end;
```

В приведенном фрагменте кода реализовано чтение текста по предложениям. По нажатию кнопки **Читать** происходит поиск точки в тексте; если она не найдена, читается весь текст, иначе — текст до следующей точки. При

этом читаемый текст удаляется из глобальной переменной `t`, содержащей текст, а также запоминается идентификатор запроса (в переменной `RID`) и включается флажок `Reading`, показывающий продолжение чтения. При наступлении события `OnRequestComplete` идентификатор запроса проверяется на равенство с контрольным значением, текст сверяется с пустой строкой, т. е. проверяется, не закончился ли текст и не нажата ли кнопка остановки чтения (не равен ли флажок `Reading` значению `True`). В случае удачного прохождения всех проверок процедура выполняется снова, пока глобальная переменная текста не станет равна пустой строке, или флаг остановки чтения не будет равен `False`.

Подобным образом возможно организовать также паузу в чтении. Для этого нужно продолжать читать текст из глобальной переменной текста, а не загружать его заново. Именно так реализовано чтение в проекте каталога 39_SpeakPause, однако здесь функция выделения предложения в тексте вынесена в отдельную процедуру, благодаря чему код несколько упрощается. Кроме того, тут обеспечено определение конца предложения не по одному символу точки ("."), а по точке с пробелом (". "), и вместо переменной для хранения состояния паузы используется целочисленное свойство кнопки паузы `Tag`:

```
btPause.Tag:=1;
```

Коррекция речи

Полагаю, наигравшись с читающим приложением, одни читатели скажут: "Фу", а другие воскликнут: "Супер! То, что надо!", иные же просто скептически промолчат. Что ж, речевые технологии далеки от совершенства, но тем не менее должны вызывать уважение оттого, что над ними трудилось огромное число лингвистов, математиков и программистов. Синтез речи можно использовать не только как игрушку в своем приложении — для людей с ограниченными возможностями (например, зрения) он очень важен, тем более что даже невысококлассные движки обеспечивают достаточно понятное чтение.

В процессе использования речи рано или поздно возникнет необходимость изменять такие параметры чтения, как тон и скорость, а также в двусмысленных фразах расставлять логические ударения (в таких, например, как классическое "казнить нельзя, помиловать!"). Очень приятно, что в синтезе речи учитываются знаки препинания.

Основным способом коррекции речи являются речевые теги, вставляемые в читаемую текстовую строку. Теги — это обычные наборы символов, составленные определенным образом, — так, чтобы сервер рассматривал их как управляющие команды, а не как читаемый текст.

Речевые теги представляют собой команды с одним или несколькими параметрами, хотя существуют теги и без параметров. Вот некоторые правила синтаксиса для речевых тегов.

- Все теги начинаются и заканчиваются символом наклонной черты влево (\) —"бэкслэшем".
- Отдельный бэкслэш не допускается внутри тега. Чтобы включить бэкслэш в текстовый параметр тега, используйте две наклонные черты влево (\\).
- Теги воспринимаются без учета регистра. Например, тег `\Spd=100\` — тот же самый, что и тег `\SPD=100\`.
- Теги восприимчивы к пробелам, т. е. не допускают внутри себя лишних пробелов. Например, тег `\Rst\` — не то же самое, что тег `\ Rst \`.

В табл. 2.18 представлены все теги, поддерживаемые MS Agent.

Таблица 2.18. *Теги, поддерживаемые MS Agent*

Тег	Синтаксис	Описание
`Chr`	`\Chr=`*`string`*`\`	Тег устанавливает тип голоса в соответствии со значениями: "`Normal`" (значение по умолчанию) — нормальный голос; "`Monotone`" — монотонный; "`Whisper`" — шепот. Кроме того, допустимые значения этого тега могут различаться для разных движков, т. е. одни движки способны поддерживать какие-то значения, а другие не поддерживают часть из них
`Ctx`	`\Ctx=`*`string`*`\`	Тег устанавливает тип содержимого читаемой строки: "`Address`" — адрес и (или) телефонный номер; "`E-mail`" — адрес электронной почты; "`Unknown`" — неопределенный тип. Значение этого тега также может варьироваться в различных речевых движках
`Emp`	`\Emp\`	Тег задает интонационное усиление следующего за ним слова. Выделяемое усилением интонации слово должно следовать сразу за тегом. Этот тег тоже поддерживается или не поддерживается в зависимости от типа Text-To-Speech генератора, поэтому данные параметры могут существовать у различных речевых синтезаторов

Таблица 2.18 *(окончание)*

Тег	Синтаксис	Описание
`Lst`	`\Lst\`	Тег указывает повторить последнее произнесенное выражение. Не поддерживается при чтении функциями API, т. е. повторяется последняя фраза при чтении методом `Speak` у персонажа
`Map`	`\Map="`*spokentext*`"="`*balloontext*`"\`	Тег указывает, как должен отображаться читаемый текст в воздушном шарике. Это удобно тогда, когда нужно прочитать один текст, а показать вместо него другой. Если необходимо поддерживать свой собственный словарь произношений в программе, то, например, для того, чтобы читать слово "юзер" как "йууузер", приходится пользоваться этим тегом, и тогда само слово в воздушном шарике будет иметь нормальный вид. Естественно, что этот тег тоже поддерживается при чтении MS Agent, а не функциями API
`Mrk`	`\Mrk=`*number*`\`	Тег определяет в тексте закладку. Затем при чтении генерируется событие, которое можно обработать (подробнее рассмотрим это позже). Значение должно быть целым числом, бóльшим 0, и не должно равняться 2 147 483 647 или 2 147 483 646
`Pau`	`\Pau=`*number*`\`	Тег определяет паузу в речи на указанное количество миллисекунд (1 с = 1000 мс)
`Pit`	`\Pit=`*number*`\`	Тег указывает тон речи в герцах. Диапазон значений зависит от речевого синтезатора (проблема определения этого диапазона будет рассмотрена в *гл. 3*)
`Rst`	`\Rst\`	Тег сбрасывает значения всех тегов на значения по умолчанию
`Spd`	`\Spd=`*number*`\`	Тег задает скорость речи (количество слов в минуту). Диапазон значений зависит от речевого синтезатора (проблема будет рассмотрена в *гл. 3*)
`Vol`	`\Vol=`*number*`\`	Тег задает громкость речи. Диапазон значений от 0 до 65 535

Теги просто вставляются в нужное место строки, читаемой методом `Speak`. Они могут следовать один за другим. В текстовом воздушном шарике теги

не отображаются, поэтому насчет этого можно не беспокоиться, зато эффект от их использования должен вам понравиться.

Приведу пример использования речевых тегов:

```
Chars.Speak('\Spd=100\Сейчас я говорю \Map="мйэдлнно"="меделнно"\.'+
        '\Spd=150\\pit=80\А сейчас быстро и низким голосом!', '');
```

Установленные при одном запуске чтения теги сбрасываются при следующем запуске метода `Speak`.

Вообще говоря, набор речевых тегов, поддерживаемых Speech API, намного шире, чем тот, что поддерживается MS Agent. Однако реальный круг доступных команд определяется выбранным синтезатором речи. MS Agent понимает самые необходимые теги, которые воспринимаются большинством синтезаторов, поэтому некоторые из них все-таки могут не дать эффекта при работе с определенными движками. Хотя сам компонент MS Agent не синтезирует речь, а пользуется Speech API, он контролирует ввод тегов для обеспечения возможности включения помимо SAPI-тегов еще и тех, которые имеют смысл именно при чтении текста анимированным персонажем, таких как `MAP` и `LST`.

Тег `MAP` определяет закладку в тексте, имеющую значение целочисленного параметра. При чтении, когда персонаж дойдет до закладки, для него сгенерируется событие `OnBookmark`, которому в виде параметра будет передано значение номера закладки. Этим удобно пользоваться в тех случаях, когда персонаж демонстрирует что-нибудь пользователю, и в ходе рассказа ему следует в определенный момент указать на что-либо рукой или воспроизвести анимацию. Обработав вышеупомянутое событие, эти действия персонажа с легкостью можно синхронизировать.

Из табл. 2.18 видно, что тег `LST` позволяет повторить предыдущий текст. Текст повторяется со всеми установленными в нем тегами, однако нужно проследить, чтобы ранее ваша программа запускала метод `Speak` с каким-либо текстом, после чего его можно будет повторять сколько угодно. Назначение этого тега очевидно: довольно часто в игровых программах реализуется повторение последней фразы, если пользователь ее не расслышал. Так вот, в этом случае для повторения нет необходимости заново строить фразу, а нужно всего лишь запустить метод `Speak` с текстом в виде тега `LST`:

```
Chars.Speak('\Lst\', '');
```

Замечание

Если последнее чтение методом `Speak` было осуществлено по принципу выбора случайной фразы из ряда возможных, разделенных вертикальной чертой "|", то в случае повторения чтения через тег `LST` повторится именно та фраза, которая была реально произнесена, т. е. она не будет выбираться случайно, как в прошлый раз.

В примере каталога 40_AgentTags (рабочее окно приложения показано на рис. 2.48) реализована демонстрация всех возможных тегов, поддерживаемых MS Agent.

Рис. 2.48. Рабочее окно приложения, демонстрирующего работу речевых тегов MS Agent

Данный проект интересен лишь как пример, поскольку в нем не реализовано чтение с паузой, в связи с тем, что при достижении ее заново вызываемый метод `Speak` для чтения каждого предложения сбрасывает все теги. На этот счет могу заметить, что в большинстве случаев скорость речи и тембр голоса меняются с помощью таких элементов интерфейса, как `TrackBar`. Поэтому вполне логично будет использовать при этом теги скорости, тембра и громкости, а для этого нужно всего лишь подставлять их (теги) автоматически при вызове метода `Speak`. Тем самым мы добьемся чтения текста кусками с сохраненными настройками голоса. Например:

```
Chars.Speak('\Spd='+IntToStr(trSpeed.Position)+'\\Pit='+
     IntToStr(trPitch.Position)+'\\Vol='+IntToStr(trVolume.Position)+
     '\'+s, '');
```

Тут элементы `trSpeed`, `trPitch` и `trVolume` — полосы изменения скорости, тона и громкости речи соответственно. Подобный подход реализован в примере каталога 41_AgentReader.

На этом можно считать завершенным знакомство с технологиями синтеза речи, поддерживаемыми MS Agent. Последнее, что нам осталось рассмотреть в данном разделе, — это распознавание речи средствами MS Agent.

Объект *SpeechInput*

Как и рассмотренный нами объект `AudioOutput`, объект `SpeechInput` обеспечивает связь со свойствами речевого ввода, реализуемого сервером.

Так же как и в объекте `AudioOutput` в объекте речевого ввода присутствуют, в основном, свойства, доступные только для чтения, причем все они служат для определения глобальных настроек, изменяемых через диалоговое окно **Дополнительные параметры персонажей** (Advanced Character Options). Все свойства объекта `SpeechInput` перечислены в табл. 2.19.

Таблица 2.19. *Свойства объекта* `SpeechInpu`

Свойство	Описание
`property Enabled: WordBool`	Показывает, доступен ли вообще речевой ввод (`True`, если доступен, `False` — иначе)
`property HotKey: WideString`	Возвращает название горячей клавиши, по нажатию на которую происходит активация распознавания речевых команд пользователя (может изменяться только пользователем)
`property ListeningTip: WordBool`	Показывает, отображается ли окно подсказки под персонажем в то время, когда он слушает команды пользователя (`True`, если отображается, `False` — иначе)

В демонстрационном проекте каталога 42_SpeechInput отражаются текущие значения этих глобальных свойств.

При рассмотрении проекта вы можете заметить еще три свойства: `Engine`, `Installed` и `Language`. Поскольку эти свойства уже не поддерживаются MSAgent v2.0, они не имеют определенного значения, т. е. возвращают значение `nil`, если запрашиваются. О свойстве, которое заменяет эти два свойства, мы поговорим уже в следующем подразделе.

Распознавание команд, или еще раз об объекте *Commands*

В *последнем подразделе разд. 2.7*, при знакомстве с использованием нескольких персонажей в нескольких приложениях мы обращали внимание на то, какой персонаж активируется для распознавания речи, являясь input-active

персонажем. Тогда мы не вдавались в подробности, самым главным для нас было понять взаимодействие наших ассистентов. Теперь же займемся тем, что обратим внимание на построение приложений, которые с помощью анимированного персонажа посредством речевых технологий Speech API распознают голосовые команды.

Мы уже работали с объектом коллекции команд `Commands` и объектами, являющимися элементами коллекции, — `Command`. С их помощью мы создавали контекстное меню персонажа. Ранее я намеренно упускал из виду свойства, связанные с голосом (Voice).

Хотя в сводных таблицах эти значения и были описаны, в тексте мы их пока не касались. Конечно, наиболее пытливые читатели скорее всего уже покопались в коде и разобрались в деталях и тонкостях, но все же некоторые пояснения я здесь приведу.

Перед тем как заняться распознаванием речи, не забудьте установить все необходимые компоненты: пакет функций Speech API, который должен у вас быть, если вы работали с синтезом речи в прошлых примерах; и движок распознавания речи (например, модуль от Microsoft для англоязычной речи (ссылка дана в *приложении 1*).

Уникальный идентификатор используемого движка можно определить по свойству `SRModeID` объекта `Character`, а также с помощью этого свойства можно изменить используемый движок распознавания речи для конкретного персонажа:

```
Chars.SRModeID:= '{D8905400-B5C8-11D0-B968020AFDB1B9C}';
```

Данной строкой кода текущий модуль распознавания речи заменяется на движок Microsoft Speech Recognition Engine v4.0, но каждый раз этого делать не надо, т. к., если в системе установлен только один модуль, то он и будет использоваться.

Кроме используемого движка распознавания речи полезно контролировать еще один немаловажный параметр, который показывает, может ли персонаж начать распознавание речи — это целочисленное свойство `SRStatus` объекта `Character`. Глагол "показывает" нужно понимать дословно, т. к. свойство предназначено только для чтения. Значения свойства `SRStatus` приведены в табл. 2.20.

Таблица 2.20. *Значения свойства* `SRStatus`

Значение	Описание
0	Ввод речи поддерживается
1	Отсутствует звуковое устройство, доступное системе для ввода голосовых данных. Естественно, что ни по значению данного свойства, ни по какому другому значению невозможно определить подключенность микрофона к звуковой карте. В данном случае значение показывает, что пользователь не имеет должным образом установленной звуковой карты с поддержкой ввода звуковой информации

Таблица 2.20 *(окончание)*

Значение	Описание
2	Данное клиентское приложение не является активным клиентом персонажа, или персонаж — не input-active персонаж (т. е. не самый верхний). Это означает, что персонаж не может воспринимать голосовые команды
3	Текущий канал ввода/вывода звука занят приложением
4	Неопределенная ошибка, произошедшая в процессе инициализации подсистемы распознавания речи. Может возникать вследствие отсутствия модуля распознавания речи соответствующего языковым настройкам персонажа
5	Речевой ввод отключен в окне дополнительных параметров персонажей
6	Неизвестная ошибка, произошедшая в ходе проверки аудиосостояния, которая не была возвращена системой

Создавать команды мы уже умеем, а что же требуется для того, чтобы эти команды начали распознаваться?

Первым делом нужно определить свойство `Voice` речевой команды, т. е. присвоить ей какое-либо значение. Это можно сделать либо при добавлении команды, либо уже после него, изменив данное свойство.

```
Chars.Commands.Add('Move', 'Переместить', 'Move to', true, true);
```

Первые да и последние два параметра мы уже знаем. Первый — имя команды, второй — подпись в раскрывающемся меню. Третий и четвертый параметры определяются свойствами `Enabled` и `Visible` самой команды. Обратите внимание на третий строковый параметр: это и есть та фраза, которую будет искать в улавливаемых звуках движок распознавания речи. В иных случаях допускается изменение свойства `Voice` в уже существующей команде:

```
Chars.Commands.Add('Move', 'Переместить', '', true, true);
Chars.Commands.Command('Move').Voice:='Move to';
```

Интересно заметить, что свойство голосовой команды может не только иметь обычные значения, но и способно принимать вид выражения типа:

```
Command.Voice:='Move [to]';
```

или:

```
Command.Voice:='Move(to|on)';
```

Подобные способы построения голосовых выражений призваны упростить создание синонимического ряда фраз, ассоциированных в одной команде.

В них допускаются следующие элементы синтаксиса, или "операторы":

- [] — квадратные скобки;
- () — круглые скобки;
- + — символ плюс;
- * — символ звездочка;
- | — символ вертикальная черта.

Выражение, находящееся в квадратных скобках (например, [*что-нибудь*]), является необязательным элементом фразы, т. е. команда, ассоциированная со следующим грамматическим выражением, будет вызываться по фразе "Move to" и "Move":

```
'Move [to]'
```

Вертикальная черта ("|") указывает на наличие альтернативы — это значит, что элементы, разделенные ею, будут носить альтернативный характер. Давайте прочувствуем это на примере.

```
'find|search text'
```

По данному выражению могут распознаваться фразы "find text" и "search text". Согласитесь, это очень удобно, поскольку не нужно вводить в коллекцию `Commands` множество одинаковых команд со слегка отличающимися частями.

Символ "звездочка" ("*") обозначает, что фраза, стоящая непосредственно перед ним, может повторяться сколь угодно большое количество раз или вообще не произноситься. Например:

```
'Please* run programm'
```

В данном примере распознавание сработает на фразы "Run programm", "Please run programm", "Please please run programm" и т. д.

Если часть фразы должна быть обязательно произнесена хотя бы один раз, то следует использовать знак "плюс" ("+"). Например:

```
'Please+ run programm'
```

Команда со знаком ("оператором") "+" исключит из рассмотрения вариант без слова "please", т. е. это слово должно быть произнесено хотя бы однажды.

Используя операторы звездочка ("*") и плюс ("+"), необходимо следить за тем, чтобы повторяемое слово стояло непосредственно перед символом оператора. Что же делать, если в роли повторяемой части фразы выступает несколько слов? В этом случае, как и в математике, используются операторные скобки:

```
'(Please run)+ programm'
```

Использование комбинаций оператора альтернативы ("|"), операторных скобок и операторов повторения може давать поразительные результаты. В до-

кументации к MS Agent приводится любопытный пример кода, с помощью которого можно организовать распознавание телефонных номеров. Естественно, что создать полный список всех комбинаций многозначных цифр через набор отдельных фраз невозможно, а вот с использованием синтаксиса построения грамматических выражений это сделать очень просто:

```
'call(one|two|three|four|five|six|seven|eight|nine|zero|oh)*'
```

Тем самым мы показываем, что каждый элемент альтернативы может повторяться сколько угодно раз либо вообще не повторяться.

Следующий прием может пригодиться тем пользователям, которые часто употребляют слова-сорняки, типа "блин", "на фиг" и т. п. Если указать в свойстве элемент "[]", речевой движок в этом месте фразы будет пропускать любые слова. Например:

```
'[...] save file [...]'
```

Под такое грамматическое выражение будут подходить фразы типа: "Блин, сэйф файл, на фиг!" или "Эй ты, save file!"

Хочется отметить, что описанными приемами нужно пользоваться с особой осторожностью и внимательно просчитывать возможные совпадения фраз и их отличия, во избежание двусмыслицы. Чем меньше похожих фраз будет в списке команд, тем больше вероятность того, что они будут успешно распознаны. Например:

```
'Save [...]'
'Save [as] text'
```

Маловероятно, что при назначении этих двух фраз разным командам эти фразы станут стопроцентно отличаться друг от друга.

Совет

Не используйте знаки препинания и специальные символы в выражении фразы свойства `Voice`, это только запутает и без того сложные алгоритмы распознавания речи.

Замечание

Разнообразие элементов грамматического синтаксиса у разных модулей распознавания речи может быть различным.

Разумеется, всегда возможны ошибки при построении фраз, однако генерация исключения зависит от речевого модуля, и если он не сгенерирует ошибку, то MS Agent ничего не скажет об этом.

Вот мы и научились строить грамматические выражения для распознавания синонимичных фраз. Теперь давайте обратим внимание на следующее обстоятельство. Если голосовые команды используются только для того, чтобы ото-

бражать их в пунктах меню (а не распознавать их), то голосовое свойство команды `Voice` вовсе не требуется определять. Минимум, что нужно, это установить свойства `Caption` и `Visible` для команды в значение `True`. Поэтому ранее с пустым свойством `Voice` наши команды не отображались в диалоговом окне **Команды голосом** (Voice commands). Как только мы определяем голосовую фразу для распознавания, команда становится речевой, при этом она отображается в окне голосовых команд с текстом `Caption`; нас даже не волнует значение свойства `Visible`, команда и так будет распознаваться.

Все же бывают случаи, когда не всегда удобно отображать в пункте меню команду под тем же названием, что и в окне голосовых команд. Поэтому любая команда — объект типа `Command` — имеет свойство `VoiceCaption` интерфейса `IAgentCtlCommandEx`, показывающее, что будет написано в окне **Команды голосом** (Voice commands).

```
(Chars.Commands.Command('Call') as
            IAgentCtlCommandEx).VoiceCaption:='Call friends';
```

Если свойства `Voice` и `VoiceCaption` имеют определенные значения, то команда в любом случае будет распознаваться и отображаться в окне голосовых команд под текстом свойства `VoiceCaption` при любом раскладе остальных свойств. В случае же отсутствия значения свойства `Voice` команда как речевая потеряет свой смысл, со всеми вытекающими отсюда последствиями.

Свойства `Voice` и `VoiceCaption` имеются не только у каждой команды объекта `Command`, но и у объекта коллекции команд (объект `Commands`, интерфейс `IAgentCtlCommandsEx`), так же как и свойство `Caption`, обозначающее подпись в меню, есть у каждой команды (у объекта `Commands` оно влияет на подпись пункта меню для переключения между группами). Аналогично, свойство `Voice` для объекта коллекции команд определяет голосовую фразу для переключения между группами команд различных клиентских приложений, а свойство `VoiceCaption` устанавливает подпись в окне **Команды голосом** (Voice commands) данной голосовой команды.

Добавлю ко всему сказанному еще один небольшой штришок. Окном голосовых команд можно управлять (в частности, его отображением) через свойство `Visible` интерфейса `IAgentCtlCommandsWindow`, указатель на который можно получить через свойство `CommandsWindow` элемента управления Agent Control. Кроме того, интерфейс поддерживает ряд других свойств, служащих для определения расположения окна и его размеров, однако эти свойства доступны только для чтения, в отличие от свойства `Visible`. В демонстрационном приложении диалоговое окно голосовых команд открывается и закрывается по щелчку на персонаже.

Мы подробно познакомились со всеми необходимыми свойствами для создания голосовых команд, и теперь для них следует написать обработчик.

Для обработки вызываемых команд пункта меню мы, как вы помните, пользовались событием `OnCommand`. В обработчике голосовых команд также следует использовать это событие. Указатель на интерфейс пользовательского ввода передается вместе с параметром `UserInput`.

```
type TAgentCommand = procedure(Sender: TObject; const UserInput: IDispatch) of object;
property OnCommand: TAgentCommand;
```

Вообще говоря, интерфейсом пользовательского ввода является интерфейс `IAgentCtlUserInput`, а не `IDispatch`, поэтому соответствующий интерфейс следует запросить через метод `QueryInterface` или привести к нему через оператор `as`. Затем, используя свойства интерфейса, можно будет определить вызванную команду, а также произнесенную голосовую фразу. Свойства интерфейса описаны в табл. 2.21.

Таблица 2.21. *Свойства интерфейса* `IAgentCtlUserInput`

Свойство	**Описание**
`property CharacterID: WideString`	Идентификатор персонажа, для которого произошло событие
`property Name: WideString`	Имя распознанной команды
`property Voice: WideString`	Распознанная голосовая фраза. Обратите внимание, значение этого свойства не идентично значению свойства `Voice` объекта `Commands` или `Command`. В данном случае оно содержит фразу, сказанную пользователем, а не выражение, заданное по синтаксическим правилам
`property Confidence: Integer`	Выражение достоверности, возвращается непосредственно речевым модулем. Возможные значения лежат в пределах от –100 до 100. Служит для лучшей идентификации голосовых команд
`property Alt1Name: WideString`	Второй вариант имени похожей команды (так называемый первый альтернативный – `Alt1`)
`property Alt1Voice: WideString`	Вариант голосовой фразы пользователя для первого альтернативного варианта
`property Alt1Confidence: Integer`	Выражение достоверности для второго варианта возможной команды (`Alt1Name`)
`property Alt2Name: WideString`	Третий вариант имени команды пользователя
`property Alt2Voice: WideString`	Голосовая фраза пользователя для третьего варианта команды

Таблица 2.21 *(окончание)*

Свойство	Описание
`property Alt2Confidence: Integer`	Выражения достоверности для третьего варианта голосовой команды
`property Count: Smallint`	Свойство показывает количество имеющихся вариантов

Теперь, определив точное название распознанной команды через свойство `Name`, мы можем создать ветвление в программном коде. Для того чтобы использовать вариантные фразы типа номеров телефонов следует прибегнуть к свойству `Voice`, содержащему распознанную голосовую фразу, произнесенную пользователем. Выделить символьный номер можно будет примерно так:

```
procedure TForm1.Agent1Command(Sender: TObject;
  const UserInput: IDispatch);
var
 cmd: IAgentCtlUserInput;
 s: string;
begin
 UserInput.QueryInterface(IID_IAgentCtlUserInput, cmd);
 if cmd.Name='Call' then
  begin
   s:=cmd.Voice;
   s:=StringReplace(s, 'one', '1', [rfReplaceAll, rfIgnoreCase]);
   s:=StringReplace(s, 'two', '2', [rfReplaceAll, rfIgnoreCase]);
   s:=StringReplace(s, 'three', '3', [rfReplaceAll, rfIgnoreCase]);
   s:=StringReplace(s, 'four', '4', [rfReplaceAll, rfIgnoreCase]);
   s:=StringReplace(s, 'five', '5', [rfReplaceAll, rfIgnoreCase]);
   s:=StringReplace(s, 'six', '6', [rfReplaceAll, rfIgnoreCase]);
   s:=StringReplace(s, 'seven', '7', [rfReplaceAll, rfIgnoreCase]);
   s:=StringReplace(s, 'eight', '8', [rfReplaceAll, rfIgnoreCase]);
   s:=StringReplace(s, 'nine', '9', [rfReplaceAll, rfIgnoreCase]);
   s:=StringReplace(s, 'zero', '0', [rfReplaceAll, rfIgnoreCase]);
   delete(s, 1, 5);
   while pos(' ', s)<>0 do
    delete(s, pos(' ', s), 1);
   ShowMessage('Номер: '+s);
  end;
end;
```

Думаю, вы заметили странное свойство `Confidence` объекта `Commands` и `Command`, а также у пользовательского ввода. Ничего особенного в нем нет — просто если присвоить этому свойству какое-то определенное значение, скажем 100, то при распознавании текста свойство `ConfidenceText` команды с наиболее близким значением `Confidence` выведется в окне подсказки `Listening Tip`, при условии, что отображение данного окна включено в окне глобальных опций. Вот пример кода создания команды:

```
Chars.Commands.Add('Move', 'Переместить', 'Move [To]', true, true);
(Chars.Commands.Command('Move') as
                  IAgentCtlCommandEx).VoiceCaption:='Move to';
Chars.Commands.DefaultCommand:='Move';
Chars.Commands.Command('Move').Confidence:=100;
Chars.Commands.Command('Move').ConfidenceText:='How do you do?';
```

При каждом успешном распознавании команды `Move` в подсказке под персонажем будет выводиться надпись с текстом "How do you do?", как показано на рис. 2.49.

Рис. 2.49. Демонстрация работы свойств `Confidence` и `ConfidenceText`

До завершения рассказа о распознавании речи средствами MS Agent совсем недалеко. Обычно для запуска процесса распознавания речи требуется нажать горячую клавишу (по умолчанию <Scroll Lock>), устанавливаемую в окне дополнительных параметров персонажей; кроме этого, можно заставить персонаж начать слушать голос программным путем с помощью метода `Listen` объекта `Character`:

```
function  Listen(Listen: WordBool): WordBool;
```

Вызывая метод с параметром `True`, мы заставляем персонаж слушать, значение `False` вынуждает его закончить "слушать" речь. Метод возвращает логическое значение, по которому можно определить, успешно ли он завершил-

ся. Так как обрабатывать речевые команды может только активное приложение, то при запуске прослушивания из неактивного метод будет завершаться неудачно, т. е. со значением `False`.

В то время как персонаж начинает слушать, сервер посылает активному приложению событие `OnListenStart` с параметром соответствующего идентификатора персонажа, а остальным клиент-приложениям — события с пустой строкой в параметре `CharacterID`.

По завершении прослушивания речи генерируется событие `OnListenComplete`, также с идентификатором персонажа (активному клиенту с идентификатором, а остальным с пустой строкой), и еще параметр, содержащий причину завершения прослушивания. Значения причины уже приводились в описании всех указанных событий, однако укажу их еще раз в табл. 2.22.

Таблица 2.22. *Значения свойства* `Cause` *в событии* `OnListenComplete`

Значение	Описание
1	Распознавание было отключено программно
2	Распознавание, включенное программным кодом, завершено по истечении тайм-аута
3	Распознавание, включенное горячей клавишей, завершено по истечении тайм-аута
4	Пользователь отпустил горячую клавишу
5	Распознавание завершено, потому что пользователь закончил говорить
6	Приложение-клиент, получающее ввод, стало неактивным
7	Персонаж по умолчанию был изменен
8	Пользователь отключил речевой ввод

Все то, о чем мы сейчас говорили, демонстрирует проект каталога 43_RecognitionCmd (рабочее окно приложения показано на рис. 2.50). Попробуйте поиграть с различными командами, подавайте похожие и смотрите, как формулировка выражения влияет на качество распознавания.

Итак, подытожим, что же нужно сделать для создания распознаваемых команд.

1. Прежде всего, конечно, установить все необходимые компоненты.
2. Выбрать требуемый голосовой движок (можно пропустить этот пункт, если движок единственный).
3. Установить нужный язык распознавания, т. к. по умолчанию у персонажа установлен язык системы.

4. Создать команду.
5. Установить речевое распознаваемое выражение (свойство `Voice`).
6. По желанию определить свойство `VoiceCaption`, чтобы в окне голосовых команд выводилась желаемая подпись команды.
7. Описать обработчик события `OnCommand` для выполнения определенных действий в качестве реакции на команды.
8. По желанию создать элемент интерфейса, который будет управлять включением прослушивания и его прерыванием.

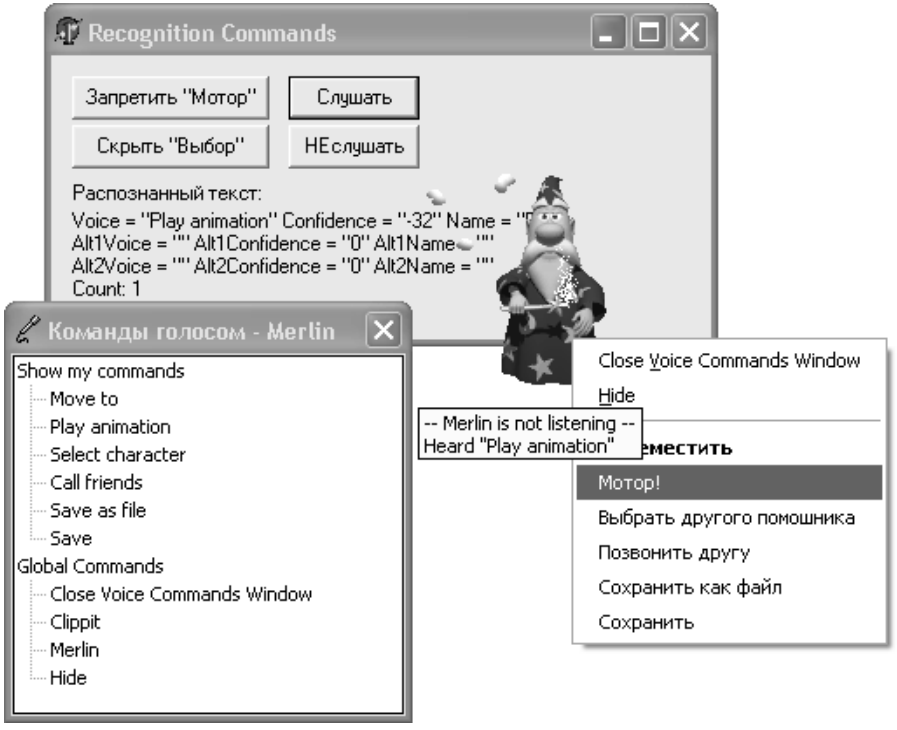


Рис. 2.50. Рабочее окно приложения, демонстрирующего распознавание команд

Безусловно, последний пункт не является обязательным, т. к. обычно бывает быстрее включить распознавание речи горячей клавишей, нежели искать окно приложения между открытыми окнами и нажимать на нужную кнопку.

Думаю, что читателям понравилось использовать голосовые команды (хотя они иногда не очень-то правильно распознаются, но все-таки!). Согласитесь, как просто устроен процесс управления речевыми командами, синте-

зом речи, да и вообще весь интерфейс разработчика MS Agent! Надеюсь, что вам пришлось по душе программирование анимированного персонажа и что материала данной главы и подборки авторских примеров оказалось вполне достаточно. В последней главе книги *(см. гл. 4)* мы снова вернемся к анимированному персонажу, чтобы объединить технологию MS Agent, Speech API и некоторые сторонние разработки (такие как BalloonDialog), смежные с MS Agent.

2.9. MS Agent на web-страницах

Наверняка даже Delphi-программистам интересно знать, каким образом можно встраивать персонаж MSAgent на web-страницы.

В нескольких строках я покажу здесь, как "встроить MS Agent в HTML", и все станет ясно, т. к. никаких изменений в использовании MS Agent не будет.

Так как элемент управления AgentControl является обычным элементом ActiveX, то даже в HTML с ним нужно работать как с ActiveX. Поэтому первое, с чего нужно начать, это встроить AgentControl на web-страницу через тег `OBJECT`:

```
<object
  id="MyAgent"
  width="0"
  height="0"
  classid="CLSID:D45FD31B-5C6E-11D1-9EC1-00C04FD7081F"
  codebase="#VERSION=2,0,0,0">
</object>
```

Самый главный параметр — `CLASID` (CLASs IDentifer) — нужно смотреть в модуле AgentObjects_TLB.pas, там есть константа, которая так и называется `CLASS_Agent`. Ее значение нужно подставить в соответсвующий параметр тега `OBJECT`.

Второй шаг — загрузка самого персонажа при загрузке web-страницы. Делается это через скрипт на любом скриптовом языке, например, на Visual Basic:

```
<script language="VBScript">
  Dim Char 'Переменная для хранения персонажа,
           'что-то вроде IAgentCtlCharacterEx

  Sub window_OnLoad 'При загрузке страницы
     MyAgent.Connected = True
        'Загрузка персонажа
```

```
        s="merlin.acs"
        MyAgent.Characters.Load "Char", s
        Set Char= MyAgent.Characters("Char")
     Char.LanguageID = &H0419 'Меняем язык на русский
     Char.Show 'Показываем персонаж
  End Sub
</script>
```

Вот уже все и проясняется. В документации к MS Agent рекомендован другой способ получения ссылки на AgentControl — через функцию `CreateObject`.

```
Function HaveAgent()
  Dim agent
  HaveAgent = False
  On Error Resume Next
  Set agent = CreateObject("Agent.Control.2")
  HaveAgent = IsObject(agent)
End Function
```

Функция возвращает значение `True` в случае успешного получения ссылки на AgentControl. Теперь вам остается лишь прописать на web-странице обработчики событий, которые будут заставлять персонаж выполнять какие-либо действия.

Обращения к методам и свойствам здесь аналогичны обращениям к ним из Delphi, при этом, разумеется, нужно учитывать синтаксис скриптового языка. Приведу пример перемещения персонажа на web-странице по щелчку на кнопке через метод `MoveTo`. Координаты нового местоположения пусть указываются в соответсвующих полях редактирования. Где-нибудь на web-странице следует вставить теги для организации элементов формы:

```
<input type="text" name="PosX" size="7" value="0">
<input type="text" name="PosY" size="7" value="0">
<input class="but" type="submit" name="MoveAgent" value="Переместить">
```

Далее нужно добавить код для обработки события щелчка мыши на кнопке `MoveAgent`:

```
<script language="VBScript">
  Sub MoveAgent_OnClick()
    Char.MoveTo PosX.value, PosY.value, 1
  End Sub
</script>
```

А следующий пример показывает реализацию чтения текста, введенного в поле редактирования, на web-странице.

Сначала описываем элементы интерфейса:

```
<input type="text" name="EditText" size="15" value="Привет!">
<input class="but" type="submit" name="ReadText" value="Сказать">
```

Соответствующий скрипт будет иметь следующий вид:

```
<script language="VBScript">
  Sub ReadText_OnClick()
    Char.Speak EditText.value, ""
  End Sub
</script>
```

Надюсь, что после внимательного изучения всего вышеизложенного вы легко сможете встраивать персонаж MS Agent на свои web-страницы. Все необходимые методы, свойства и события AgentControl мы уже подробно рассмотрели.

2.10. Резюме

Трудно из материала большой главы выделить самое главное, однако я попытаюсь это сделать.

Наиболее важное, что мы должны понять, — Microsoft Agent есть компонент операционной системы, представляющий собой OLE Automation сервер, обеспечивающий функционирование анимированного персонажа.

Кроме самого компонента, для реальной работы службы MS Agent требуются персонажи, распространяемые в файлах структурного формата ACS (описание агента и данные анимации в одном файле) и в раздельном формате, где описание находится в файле ACF, а данные анимации в файлах ACA, по одной анимации в каждом файле.

Вследствие дробления данных анимаций на части вторая форма нашла большее распространение в сети Интернет, поскольку ее применение экономит время загрузки данных анимаций, т. к. обычно абсолютно все возможные анимации не используются.

Нами рассмотрены рекомендации по программированию анимированного персонажа. Из них можно заключить, что главный упор разработчик приложения должен делать на существование альтернативы практически во всех аспектах. Рекомендуется не навязывать пользователю анимированного агента (возможно обходиться и без него), дублировать речевой синтез другими источниками информации, например, обычными сообщениями, и полностью не полагаться на распознавание речи. Ну и, конечно, надо стараться, насколько возможно, "очеловечивать" виртуального помощника, подобирая синонимический ряд фраз и анимаций, коль скоро используется синтез речи. Следует подготовить удобоваримые, быстро запоминающиеся и мало

похожие друг на друга голосовые команды в случае применения распознавания речи.

MS Agent обладает собственной моделью объектов, связь между которыми мы уже рассматривали. Не следует эти объекты воспринимать как экземпляры какого-либо класса, представлять их себе надо как наборы интерфейсов со своими свойствами и методами, взаимодействующими друг с другом внутренними способами, реализованными в сервере MS Agent.

Компонент главным образом включает в себя сервер Agnet Server и элемент управления Agent Control. Естественно, что можно и напрямую обращаться к серверу MS Agent, однако иногда это не очень удобно. Поэтому обычно работают с тем элементом управления, с которым удобнее это делать, как с COM-объектом, создавая экземпляр объекта через библиотеку COM, а также и через интерфейсы OLE Automation. Так как мы работаем в среде Delphi, то еще одним способом работы можно назвать использование элемента управления Agent Control как обычного компонента в визуальной среде программирования.

Всю деятельность агента легко свести к воспроизведению определенной анимации. Действительно, все время персонаж только тем и занимается, что прокручивает какую-либо анимацию. Группы отдельных анимаций определяют анимации состояния агента, т. е. в определенные моменты времени персонаж самостоятельно может воспроизводить заданную его создателем анимацию, к примеру такую, какая ассоциирована с состоянием безделья.

Нужно понимать, что практически все действия, совершаемые анимированным персонажем, асинхронны с вызовом его методов, т. е. их последовательность образует очередь, которая выполняется в определенный промежуток времени. Для определения состояния выполнения запроса следует использовать объект `Request`, который создается сервером при вызове подобных методов и возвращает указатель на него через методы.

Прерыванием или остановкой действий или групп действий, стоящих в очереди конкретного персонажа, можно управлять через определенные методы, такие как `Intrrupt`, `Stop` и `StopAll`.

При использовании персонажа раздельного формата через web-сервер перед первым использованием анимации следует отыскивать необходимую анимацию или анимации состояния через метод `Get`, иначе ожидаемого эффекта не будет. Однако применение раздельного формата персонажа ничем не отличается от использования обычного формата ACS, если персонаж установлен на локальном компьютере (не через протокол HTTP).

Для использования речевых технологий вместе с анимированным персонажем следует первым делом установить необходимые компоненты, т. к. сам MS Agent ничего подобного делать не умеет, а лишь реализует простейший программный интерфейс для конечного разработчика. Такими компонента-

ми являются: пакет функций Speech API, модули синтеза и распознавания речи, иначе называемые речевыми движками. Использование данных функций средствами API — это отдельная тема для разговора, и ее мы рассмотрим уже в следующей главе.

Распознавание речи поддерживается MS Agent в виде распознавания отдельных команд, поэтому для того, чтобы речевая реплика имела возможность распознаваться и ассоциироваться с запуском определенных действий, нужно соответствующим образом указать контрольную фразу, так сказать оригинал, и прописать обработчик события.

Вот, пожалуй, и все. Надеюсь, что вам доставило удовольствие знакомство с виртуальным миром агентов, и вы также охотно познакомитесь с остальными частями книги. Кроме того, можете обратиться к англоязычной документации Microsoft Agent SDK для получения более полной информации, но мне кажется по-русски и с развернутыми примерами все-таки читать лучше.

Не хочу, чтобы отсутствие некоторых компонентов на прилагаемом к книге компакт-диске и необходимость их скачивания из Интернета стали причиной вашего недовольства и потерей желания знакомства с технологией MS Agent. Мне кажется, она того стоит, тем более, что компоненты бесплатны, хотя и имеют сложную лицензию на их использование и распространение.

Глава 3

Технология Speech API в среде Delphi

Данная глава книги посвящена речевым технологиям, реализуемым в среде программирования Borland Delphi. Ранее, при рассмотрении вопросов создания приложений, включающих Microsoft Agent, мы уже касались функций, которые реализуют преобразование текста в связную речь, а также распознавание голосовых команд и определение реакций на них. Однако элемент управления MS Agent не позволяет управлять синтезом речи или ее распознаванием на более низком уровне; к счастью это или к сожалению — судить вам.

Не сомневаюсь, что читателям будет интересно познакомиться с некоторыми разделами технологии Speech API. Так, например, прочитав и поняв материал главы, вы научитесь создавать программы, синтезирующие речь, регулировать параметры речи, записывать выводимую речь в звуковой файл, как это делают многие известные программы. Кроме того, вы сможете самостоятельно написать приложение, распознающее речь, понимающее речевые команды и диктовку текста. Попутно вы узнаете, как производится поиск установленных в системе речевых синтезаторов, как для распознавания, так и для синтеза речи, а также каким образом определяются их основные рабочие параметры.

Эта глава может кому-то показаться самой сложной, т. к. здесь будет намного меньше рисунков, чем в предыдущих главах. Да и для понимания материала уже недостаточно будет всего лишь взглянуть на программный код в тексте. Если в главе об MS Agent *(см. гл. 2)* кое-кто мог, рассмотрев скриншот, сразу понять, о чем идет речь, а запустив приложение, уже окончательно разобраться в вопросе, то здесь нужно внимательно разобрать предложенные примеры, поиграть с ними, проследить за некоторыми значениями в процессе отладки, чтобы уяснить смысл различных параметров и свойств. Ведь на рисунке не покажешь процесс распознавания речи или синтез.

Прежде чем приступить к программированию, мы кратко познакомимся с предысторией создания и развития этой молодой технологии.

Итак, если хотите, чтобы Delphi заговорил — читайте эту главу!

3.1. Введение в речевые технологии

Речевые технологии — технологии XXI века. Благодаря им появилась возможность управлять компьютером с помощью голоса, диктовать текст, слушать, а не читать книги, а в перспективе — общаться с компьютером на интеллектуальном уровне. Основными направлениями исследований в этой области являются: распознавание речи, синтез голоса, средства речевого управления, идентификация по образцу речи.

Попытаюсь поточнее обозначить сферы применения речевых технологий.

О чем это мы?

Еще в 50—60-е годы прошлого века многие надеялись, что человек будет общаться с компьютером, обращаясь к нему исключительно с помощью собственного голоса всего через каких-то 10—15 лет. Затем "бесклавиатурное" общение отодвинулось на конец 80-х годов, потом и 90-х. В наше время в решении этой задачи наметился заметный прогресс. Программы и системы, обладающие средствами речевого ввода информации, получают все большее распространение, хотя они еще далеки от совершенства. Проблема оказалась весьма сложной, обладающей свойством разветвляться в другие области знаний: статистическую радиотехнику, лингвистику и т. д.

На сегодняшний день принято считать, что существует две основные речевые технологии — распознавание речи и синтез речи. Одна обеспечивает ввод речевой информации в компьютер, другая — вывод.

Несмотря на успехи последнего времени, некоторые специалисты скептически относятся к возможности реализации речевого интерфейса в ближайшие годы, хотя кое-кто и считает, что эта задача практически решена.

Принято полагать, что построение речевого интерфейса предусматривает необходимость реализации трех задач.

- Первая задача состоит в том, чтобы обеспечить восприятие компьютером отдельных команд. Пока речевое сообщение состоит из определенного стандартного набора понятных компьютеру речевых команд (например, дублирующих пункты меню), сложностей в решении задачи нет. Хотя такой способ общения с компьютером и не будет намного удобнее, чем ввод этих же команд при помощи мыши или с клавиатуры. Пожалуй, удобнее щелкнуть мышкой на ярлыке приложения, чем четко выговаривать "Пуск! Главное меню! Total Commander!" А ведь еще хотелось бы, чтобы компьютер безошибочно понимал естественную речь человека и воспринимал, к примеру, слова "Конец!" и "Ну все, заканчивай!" как одну и ту же команду.

- Вторая задача заключается в распознавании компьютером слитной речи человека в целях полной замены использования клавиатуры при наборе текста на диктовку.
- Третья задача — обеспечить способность компьютера преобразовывать информацию, с которой он оперирует, в речевое сообщение, понятное человеку.

Из перечисленных задач совершенно ясное решение существует только для последней. По сути, синтез речи — это чисто математическая задача, которая в настоящее время решена на довольно хорошем уровне. И в ближайшем будущем, скорее всего, будет совершенствоваться только ее техническая реализация.

Решению первой задачи препятствует то обстоятельство, что до сих пор не выяснено, каким образом можно расчленить нашу речь так, чтобы извлечь из нее те составляющие, в которых содержится смысл. В том звуковом потоке, который мы выдаем при разговоре, нельзя различить ни отдельных букв, ни слогов. Однако после необходимой тренировки современные системы распознавания речи работают довольно неплохо и ошибаются не больше, чем делали оптические системы распознавания печатных символов еще несколько лет назад.

Пока же речевой интерфейс может служить лишь для дублирования голосом команд, которые могут быть введены с клавиатуры или при помощи мыши. Здесь его преимущества сегодня еще сомнительны. Правда, есть область, которая многим кажется очень привлекательной. Это речевой ввод текстов в компьютер. В самом-то деле, зачем стучать по клавиатуре, если намного удобнее продиктовать все компьютеру, чтобы он сразу записал услышанное в текстовый файл. Тут даже не требуется, чтобы машина "осмысливала" услышанное, а задача перевода речи в текст более или менее решена. Недаром большинство выпускаемых ныне программ "речевого интерфейса" ориентированы именно на ввод текста с помощью микрофона.

Вот только и тут не все так гладко, как хотелось бы. Если мы примемся читать текст вслух, четко выговаривая слова, с паузами, монотонно, как это требуется для системы распознавания речи, то на одну страницу уйдет минут пять, а большинство пользователей печатает на клавиатуре примерно с той же скоростью. То есть особого прогресса не наблюдается.

Наконец, сложившаяся практика применения компьютеров пока не способствует активному внедрению речевого интерфейса.

Для подачи команд, связанных с позиционированием в пространстве, человек всегда пользовался и будет пользоваться жестами, т. е. системой "руки-глаза". На этом принципе построен современный графический интерфейс. Ближайшая перспектива замены клавиатуры и мыши блоком распознавания речи полностью отпадает. При этом выигрыш от возложения на него части

функций управления настолько мал, что не смог предоставить достаточных оснований даже для пробного внедрения в массовых компьютерах.

Часто к достоинствам речевого ввода текста относят отсутствие необходимости в предварительном обучении. Однако одно из самых слабых мест современных систем распознавания речи — чувствительность к четкости произношения — приводит к потере этого, казалось бы, очевидного преимущества. Печатать на клавиатуре оператор учится в среднем 1—2 месяца. Постановка же правильного произношения может занять несколько лет.

Существует и еще одно досадное ограничение сферы применеия столь, казалось бы, привлекательной технологии. Пользователь, взаимодействующий с компьютером через речевой интерфейс, вынужден работать в отдельном помещении либо пользоваться звукоизолирующим шлемом. Иначе он будет мешать работе своих соседей по офису, которые, в свою очередь, создавая дополнительный шумовой фон, будут значительно затруднять работу речевого распознавателя. Таким образом, речевой интерфейс вступает в противоречие с организационной структурой предприятий, ориентированных на коллективный труд. Ограничения применимости систем распознавания речи в рамках наиболее популярных традиционных приложений заставляют сделать вывод о необходимости поиска потенциально перспективных для внедрения речевого интерфейса приложений за пределами традиционной офисной сферы, что подтверждается коммерческими успехами узкоспециализированных речевых систем. Самый успешный на сегодня проект коммерческого применения распознавания речи — телефонная сеть фирмы AT&T. Клиент может запросить одну из пяти категорий услуг, используя любые слова. Он говорит до тех пор, пока в его высказывании не встретится одно из пяти ключевых слов. Эта система в настоящее время обслуживает около миллиарда звонков в год.

Несмотря на то, что одним из наиболее перспективных направлений для внедрения систем распознавания речи может стать использование их в сфере компьютерных игр, узкоспециализированных реабилитационных программ для инвалидов, телефонных и информационных систем, ведущие разработчики речевого распознавания наращивают усилия по достижению универсализации и увеличения объемов словаря даже в ущерб сокращению процедуры предварительной настройки на диктора.

Будущее речевого интерфейса в не меньшей степени зависит от умения современных исследователей и разработчиков не только создать технологическую основу речевого ввода, но и гармонично слить технологические находки в единую логически завершенную систему взаимодействия "человек-компьютер".

Речевые технологии в программном обеспечении с каждым днем находят новые сферы применения. Начав с простого преобразования текста в речь (text-to-speech), сейчас они заняли прочное место в компьютерном мире.

Из существующих новейших направлений развития программного обеспечения в области речевых технологий можно выделить несколько основных групп:

- управление программой голосовыми командами;
- диктовка текста;
- идентификация по образцу речи;
- синтез речи;
- офисный голосовой интерфейс;
- мультимедийные обучающие системы.

Существуют также и другие категории деления.

Что напишем, то и слышим

Технология синтеза речи обеспечивает преобразование любого хранимого в компьютерном формате текста в синтетическую речь, звучащую приблизительно как человеческая. Центральным компонентом этой технологии является аппаратный синтезатор речи.

Речевой вывод используется на компьютерах уже в течение некоторого времени. Однако "репертуар" первых систем был сильно ограничен. Часто в них использовалось меню, с помощью которого пользователь выбирал, какие из предварительно записанных высказываний он хотел бы слышать. Предложенный машине для воспроизведения текст сравнивался с имеющимися в базе данных записями, и если находились подходящие образцы, они озвучивались. Этот подход широко применяется в устанавливаемых в гостиницах телефонных "будильниках", а также в автомобильных навигационных системах.

Более полные функционально средства, рассчитанные на универсальное применение в офисе, появились позднее.

Французская компания Elan Informatique разработала программу синтеза речи ProVerbe, которая позволяет осуществлять удаленный доступ к электронной почтовой корреспонденции по телефону. Английская фирма First Byte предлагает систему Monologue, предназначенную для вычитки текста. Пользуясь ею, корректор избавляется от необходимости прочитывать текст глазами. Шведский продукт Infovox синтезирует речь на 12-ти различных языках и диалектах: американском и британском английском, датском, голландском, финском, французском, немецком, исландском, итальянском, норвежском и, естественно, шведском.

Сегодня рынок предоставляет широкий выбор синтезаторов речи, большинство из них укомплектовано собственным текстовым процессором. В основном синтезаторы говорят на американском английском, но некоторые спо-

собны и к другим языкам. Иногда можно выбирать и голос: нормальный мужской, нормальный женский, басовитый мужской и т. д.

Текст превращается в речь

Существуют некоторые аналогии между вычислительной обработкой звука в процессе речевого ввода и в процессе речевого вывода. Поставщики применяют различные алгоритмы, однако базовые рабочие процессы, используемые всеми необходимыми модулями, в том числе: *нормализации*, *преобразования*, *анализа* и *обработки звука* остаются одинаковыми.

Задача первого этапа речевого синтеза состоит в том, чтобы избавиться от "шума": кавычек, скобок, апострофов и знаков препинания. Она решается модулем нормализации, который, кроме того, обрабатывает зависящие от конкретного языка сокращения, форматы дат, времени, денежных единиц, телефонных номеров и других специальных обозначений.

Модуль преобразования переводит текст из орфографического в фонетический формат (т. е. из букв в звуки). Для некоторых языков, таких как, например, немецкий, это достаточно легко, поскольку правила преобразования просты. В английском же, напротив, эти правила отличаются большой сложностью, а для многих случаев их и вовсе не существует. Например, окончание ough может произноситься шестью различными способами. В русском языке соответствующие правила довольно просты, но тоже имеются некоторые аномалии (такие как непроизносимая буква "й" в слове, пожалуйста, например). Способ воспроизведения безударного "о" (например, в слове "кто-то") может служить еще одним примером трудности синтезирования речи на компьютере.

Модуль анализа выполняет одновременно лексикографическую и синтаксическую обработку для выбора между возможными вариантами произношения, а также ритма и интонации. Лексический анализ применяется для выяснения значения слова с учетом контекста, а синтаксический — для проверки порядка следования слов с целью расстановки акцентов.

Результатом работы модуля анализа является фонетическое представление исходного текста, которое передается на вход *фонетического модуля*. Здесь данные из входного потока заменяются такими элементами речи, как дифтонги, трифтонги и четырехзвучия. Все они были выделены из естественной человеческой речи, так что в них сохранены межфонемные переходы (гладкие переходы человеческого голоса от произнесения одного звука к произнесению другого). Связывание или сцепление вместе этих речевых элементов позволяет получить высококачественную синтезированную речь. Объем требуемой для хранения отдельных образцов речи компьютерной памяти составляет примерно от 1 до 20 Мбайт, в зависимости от языка, конкретного приложения и потребностей рынка, на который оно ориентировано.

Кроме того, фонетический модуль обеспечивает разбиение текста на сегменты для формирования ритмического и интонационного рисунка (просодии). Обычно в литературных кругах понятие просодии связывается со стихотворным размером, но в контексте компьютерного синтеза речи под этим словом подразумевается ритмический и интонационный баланс предложения. Для формирования легкой для восприятия и естественно звучащей речи хорошая просодия имеет очень важное значение. Она достигается выбором надлежащей длительности воспроизведения каждой фонемы и обеспечением гладкости огибающей звуковой последовательности.

Модуль *обработки звука* преобразует фонетические данные в слышимые звуковые сигналы. Он генерирует волновые последовательности с частотой дискретизации, как правило, около 10 кГц. На этой стадии осуществляется управление громкостью, быстротой речи и высотой тона голоса.

Как это делается?

Над технологиями распознавания речи работает множество компаний, однако все они в основном следуют одной и той же базовой методологии, основанной на четырехэтапной процедуре. Различия сводятся к применяемым методам математического моделирования и алгоритмам. Один из ведущих поставщиков таких механизмов — фирма Learnout & Hauspie. Именно ее продукт использован для составления следующего описания. Система состоит из таких модулей:

- препроцессор;
- экстрактор;
- компаратор;
- интерпретатор.

Препроцессор, или модуль сбора данных, обеспечивает связь между аналоговым внешним миром и цифровым микрокосмосом компьютера. В число видов обработки входного сигнала могут входить: автоматическая регулировка усиления, подавление эха, обнаружение присутствия/отсутствия речи и обнаружение интонационного конца фразы. Цель всей этой обработки состоит в том, чтобы передать на следующий этап как можно более качественный сигнал.

Экстрактор — выполняет частотный анализ сигнала. Акустически-фонетический поток данных разбивается на короткие кадры, или векторы, продолжительностью, как правило, около 10 мс. Эти алгоритмы обладают достаточной акустической устойчивостью благодаря адаптивности канала записи и подавлению шумов на предыдущем этапе.

Компаратор — осуществляет акустическое сравнение: каждый кадр, или вектор, сравнивается с имеющимися акустическо-фонетическими образцами (субсловами). Для каждой пары вычисляется функция правдоподобия. Имен-

но в этом модуле наука ближе всего подходит к превращению ее в реальную производительную силу. Речь не отличается высоким уровнем воспроизводимости, и создателям компаратора приходится пытаться воспроизвести работу человеческого мозга, который использует различные дополнительные признаки и контекст для интуитивного определения смысла сказанного.

Механизм, разработанный специалистами Learnout & Hauspie, использует три вида субслов: контекстно-независимые фонемы, контекстно-зависимые фонемы и модели слов. Такое разнообразие элементов отражает возрастающую степень детализации моделирования.

Интерпретатор — решает задачу динамического программирования с целью нахождения наилучшего разбиения полученного от компаратора "алфавитного" потока на слова и фразы. В зависимости от объема используемого словаря и действующих синтаксических правил, применяются различные стратегии поиска и отсева.

Итак, механизм распознавания речи завершил свою работу и выдал поток слов и предложений. Чтобы воспользоваться этим его конечным продуктом из некоторого приложения, необходим интерфейс прикладного программирования (API). API дает разработчикам средства управления механизмом распознавания. В частности, он охватывает такие административные функции, как перемена языка, смена пользователя или рабочей среды, запуск или остановка процесса распознавания, выборка результатов. Возможно применение отдельных версий интерфейса для различных аппаратных платформ.

Интерфейс прикладного программирования механизма Learnout & Hauspie позволяет выбирать те или иные словарные массивы, управлять контекстами, пополнять словари фонетических исключений и организовывать распознавание речи нескольких пользователей. Существуют специальные средства проектирования, предназначенные для специализации и настройки базового механизма.

Компьютер управляется голосом

Существующие системы распознавания речи можно классифицировать по разным признакам.

- По назначению:
  - командные системы;
  - системы диктовки текста.
- По потребительским качествам:
  - дикторооориентированные (тренируемые на конкретного диктора);
  - дикторонезависимые;

- распознающие отдельные слова;
- распознающие слитную речь.

❒ По механизмам функционирования:

- простейшие (корреляционные) детекторы;
- экспертные системы с различным способом формирования и обработки базы знаний;
- вероятностно-сетевые модели принятия решения, в том числе нейронные сети.

Первыми появились именно средства речевого управления. Цель их создания заключается в том, чтобы обеспечить такое взаимодействие человека и компьютера, при котором человек обращается к компьютеру с определенными (краткими, содержащими простые команды) словами, а тот их понимает и выполняет требуемые действия. Наибольшей популярностью приложения этой группы пользуются у различных телефонных служб. Автоматические справочные системы используются многими телефонными компаниями в Европе и Америке.

Распознавание чисел можно считать на сегодняшний день уже зрелой технологией. Компьютеры способны распознавать слова, обозначающие числа, на 20-ти различных языках. Правда, необходимо, чтобы пользователь называл цифры последовательно одну за другой и "входил в положение" компьютера, учитывая испытываемые им трудности.

Следующим шагом после распознавания цифр стало распознавание имен. Некоторые способные на это системы уже используются компаниями в их внутреннем обиходе. На практике возникают некоторые сложности, связанные с особенностями речевого поведения человека. Например, в России системе придется искать в своем каталоге Петра, когда у нее попросят Петю, или Александра вместо Саши или Шуры.

Второе поколение программных продуктов было основано на применении ключевых слов. Их ограниченность обусловлена тем, что даже самую простую мысль человек способен выразить бесчисленным множеством способов. Если попытаться заложить в компьютер всю необходимую для их обработки информацию, он просто с ней не справится.

Сегодня уже существуют системы распознавания речи, принадлежащие к третьему поколению, основанные на идее обучения. В течение предварительного периода они обучаются на большом количестве диалогов. В результате этого обучения строятся рабочий словарь и база данных отношений между отдельными словами.

Долой клавиатуру?

Программы диктовки применяются на практике вот уже несколько лет. Пакет VoiceType корпорации IBM неплохо зарекомендовал себя в качестве средства для раздельной диктовки. А фирма Dragon Systems первой создала систему, способную воспринимать слитную речь. Системы раздельной диктовки проще в разработке и предъявляют более умеренные требования к мощности вычислительного устройства, однако они требуют от пользователя несколько неестественного способа произношения, с короткой паузой перед каждым следующим словом. Средства распознавания слитной речи требуют более быстрых процессоров и больших объемов памяти, но зато поток слов может изливаться из пользователя безостановочно.

Продукт Dragon Systems, который называется DragonDictate, позволяет непосредственно надиктовывать текст в программы Word, WordPerfect, Netscape Navigator, Internet Explorer и многие другие популярные приложения. Пользователь может управлять ПК голосом, в частности — позиционировать курсор мыши с абсолютной точностью. Предусмотрена последовательная система меню и команд для единообразного управления любыми приложениями, что существенно упрощает работу.

Активный словарь системы насчитывает по крайней мере 30 тыс. слов, а с учетом всех фонетических и языковых моделей — фактически вчетверо больше. Кроме того, пользователь может дополнить активный словарь многими тысячами нужных ему слов, также выпускаются готовые дополнительные словари по различным профессиональным тематикам. Особенно популярны медицинский и юридический дополнительные словари.

Общее правило состоит в том, что чем слово длиннее, тем оно легче распознается. В число самых коротких слов, относящихся в то же время и к наиболее важным, входят буквы алфавита. Отдельные буквы важно распознавать еще и потому, что с их помощью осуществляется ввод редких или необычных слов, отсутствующих в словаре системы. Если последовательность букв не складывается в известное слово, запускается алгоритм корректировки.

IBM разработала собственную систему распознавания слитной речи — ViaVoice. С ее помощью многие виды работ на компьютере можно выполнять в речевом режиме. Например, надиктовывать текст непосредственно в Windows-приложения, включая обычные письма, отчеты и электронные почтовые отправления, открывать и закрывать компьютерные файлы, а также ориентироваться в пределах "рабочего стола". Такие речевые команды, как file print, file save или scroll up/down, полностью заменяют соответствующие последовательности действий с использованием клавиатуры или мыши. Скорость ввода текста достигает 140 слов в минуту. Это намного быстрее, чем средний человек способен набирать с клавиатуры. Кроме того, функция синтеза речи позволяет компьютеру для контроля повторять вве-

денный текст вслух. В настоящее время ViaVoice существует в версиях для американского английского, британского английского, французского и немецкого языков. Начинать диктовать можно сразу же после короткого — в три предложения — курса обучения системы.

Предъявите... голос

Задача распознавания речи состоит в том, чтобы понять, что человек говорит, тогда как задача идентификации по образцу речи — убедиться, что говорящий является именно тем, за кого себя выдает, либо идентифицировать незнакомого человека. Идентификация по образцу речи тесно связана с биометрическими технологиями, предназначенными для идентификации людей по их уникальным физическим признакам, таким как отпечатки пальцев или рисунок радужной оболочки глаза. Речь, подобно подписи, определяется множеством постоянных физических параметров и, кроме того, находится под влиянием текущего состояния и поведения человека. С точки зрения конечного пользователя, эта технология отличается переносимостью, оперативностью и дешевизной... Голос невозможно потерять, украсть или взять взаймы.

Все это справедливо, однако голос все же может быть записан и воспроизведен. Правда, при этом изменяются некоторые его тона. Тем не менее, определенная возможность для введения системы в заблуждение имеется, и создатели лучших из существующих продуктов предусматривают специальные меры против подобного мошенничества.

Процесс взаимодействие конечного пользователя с системой идентификации разбивается на три фазы: регистрации, тестирования и допуска. В процессе регистрации запоминаются особенности голоса пользователя и формируется так называемая речевая модель. При тестировании выполняется сравнение предложенного образца речи с хранящейся в памяти компьютера речевой моделью пользователя, а также с моделью "самозванца", составленной на базе голосов множества других людей. Если результат сравнения окажется положительным для первого случая и отрицательным для второго, считается, что тестирование прошло успешно. Идентификацию по голосу можно использовать и в сочетании с другими средствами обеспечения безопасности.

Применяются различные подходы к организации регистрации и тестирования. Проще всего использовать для идентификации одно ключевое слово. Оно должно быть повторено в процессе регистрации два или три раза. Такая схема отличается простотой и оперативностью, хотя пользователю приходится запоминать пароль. Однако есть некоторая доля риска, что систему удастся обмануть с использованием записи.

Несколько большее время занимает процедура регистрации, состоящая в повторении десятка или более предложенных системой слов. Они могут быть разными для каждого пользователя. В процессе тестирования человеку предлагается произнести несколько слов из этого набора. Для каждого сеанса регистрации используются различные слова, что обеспечивает высокую степень защиты от мошенничества с записью. Тестирование в этом варианте также состоит в сравнении с моделями пользователя и "самозванца".

Наивысший уровень безопасности обеспечивает фоновая схема регистрации. При создании своей речевой модели пользователь зачитывает предложенный ему текст или последовательность фраз. Тестирование выполняется непрерывно во все время беседы с компьютером, в которую вкрапляются некоторые записанные элементы. В остальном тестирование выполняется по обычной схеме. Эта процедура остается для пользователя "невидимой".

О некоторых речевых программах

Среди отечественных программ, использующих синтез речи, хотелось бы назвать известную многим "Говорилку" (Govorilka, **http://www.vector-ski.ru/vecs/govorilka**). Это небольшая программа для чтения текстов голосом. Она может прочитать вслух любой текст на любом языке, любым установленным голосом, запишет текст в MP3-файл, продублирует голосом слова, которые вы набираете в любых программах.

Популярен также "Говорун+" (**http://govorunplus.narod.ru**), программа, основанная на технологии MS Agent 2.0 и использующая для синтеза речи любые SAPI-движки.

Из программ распознавания речи наиболее известны IBM ViaVoice Gold, Philips FreeSpeech 2000, L&H Voice Xpress Pro, Dragon Naturally Speaking Personal Edition и "Горыныч".

Все они рассчитаны, в первую очередь, на то, чтобы обеспечить ввод в документ текста. Первые четыре позволяют вводить английскую речь, и только последняя — русскую.

Диктовать сплошной текст во всех системах достаточно просто. Однако вносить исправления в текст с помощью голосовых команд гораздо сложнее.

Все пакеты обеспечивают диктовку непосредственно в окне любой программы, работающей с текстами, в частности, Microsoft Word, Excel, и таких популярных почтовых клиентах, как Outlook Express и Netscape Messenger. Правда, на компьютерах относительно старых моделей обработка произносимых человеком слов может происходить с задержкой.

Вы можете не только диктовать программам, но и давать им указания, например: открыть такой-то файл, напечатать такую то страницу, а также управлять перемещениями по рабочему столу.

Приведу краткий перечень достоинств и недостатков перечисленных выше программ распознавания речи.

- Dragon Naturally Speaking Personal Edition v.1.0 (**http://www.naturalspeech.com**).

  Как утверждает Dragon Systems Inc. — это первая в мире система распознавания беглой речи (создана в 1997 году). То есть с ее помощью можно набирать тексты на компьютере, просто диктуя их так, как вы обычно говорите — с живой интонацией, не разделяя слова.

  Достоинства: высокая безошибочность распознавания, простота использования.

  Недостатки: неудобный ввод чисел, посредственное управление экраном.

- IBM ViaVoice Gold (**http://www.software.ibm.com**).

  Достоинства: хорошее распознавание речи, усовершенствованное перемещение по экрану и улучшенный дизайн.

  Недостатки: низкое качество распознавания имен собственных и сокращенных слов, медленная работа в среде некоторых программ.

  По качеству распознавания текста данный пакет конкурирует с продуктом фирмы Dragon, но по уровню распознавания команд и удобству интерфейса превосходит его.

- L&H Voice Xpress Professional (**http://www.lhs.com**).

  Достоинства: простое и быстрое исправление неправильно воспринятых слов, великолепное распознавание чисел.

  Недостатки: "придирчивость" к произношению команд, неравномерное качество распознавания.

  Этот пакет содержит словарь объемом 230 тыс. слов и выражений. С его помощью пользователь может как надиктовывать текст и выполнять различные команды, так и "заставлять" компьютер производить некоторые действия по ключевому слову (команде). Процедура обучения в Voice Xpress Pro по сравнению с другими пакетами самая продолжительная.

Кроме описанных выше программ, есть множество других, работающих примерно так же. Есть и системы, предназначенные только для управления компьютером, такие как YeSir Voice Commander, IBM ViaVoice, Voice Pilot.

Подобные программы и программы для диктовки позволят вам довольно быстро (на соответствующей технике и при соответствующей тренировке) составлять и отправлять, например, электронные письма или другие документы. Однако у них есть масса недостатков. Один из них — невозможность восприятия беглой речи.

3.2. Основы технологии Speech API

Прежде чем приступить к программированию, рассмотрим некоторые компоненты, которые нам понадобятся. В этом разделе будет описан минимальный набор компонентов для работы с речью, а также процесс их установки. Затем мы перейдем к описанию объектов модели Speech API.

Необходимые компоненты и их установка

Как и любые функции, речевые технологии подразумевают наличие в операционной системе необходимых компонентов, которые обеспечили бы выполнение эксклюзивных задач. Кроме того, должен быть налажен доступ к ним из любых клиентских приложений. Как известно, подобная проблема решается с помощью COM-технологии, поэтому созданные группы COM-объектов реализуют различные задачи по запуску программ распознавания и синтеза речи.

Чтобы воспользоваться речевыми функциями, в операционной системе должен быть установлен пакет речевых функций Microsoft Speech API (SAPI). Работать мы будем с пакетом версии 4.0, хотя Windows 2000/XP включает в стандартную поставку SAPI v5.1. Может быть, по каким-либо параметрам эта версия и превосходит предыдущую, однако в ней нарушена совместимость снизу вверх. То есть речевые движки, а также программы, пользующиеся Speech-функциями Speech API v4.0, перестают работать в среде в SAPI v5.1. Правда, ничто не мешает иметь в системе пакеты речевых функций разных версий: даже при одновременном использовании разными приложениями они прекрасно уживаются. Основанием для использования четвертой версии SAPI является еще и то, что большинство отечественных речевых программ рассчитаны именно на нее. Наконец, для Speech API v4.0 существует множество модулей синтеза речи для большинства языков (в том числе и для русского).

Новый пакет речевых функций Microsoft Speech API v5.1 некоторым образом отталкивает еще и большим размером установочного файла — около 68 Мбайт (правда, в ней уже присутствует несколько синтезаторов, к сожалению, только английских, а также модуль распознавания речи), в то время как пакет функций версии 4.0 имеет дистрибутив размером всего около 824 Кбайт (правда, в дополнение к нему придется скачивать речевые движки). Дистрибутив доступен по адресу: **http://activex.microsoft.com/activex/controls/sapi/spchapi.exe**.

Установив пакет функций, мы не сможем сразу воспользоваться речевыми функциями, т. к. сам Speech API не осуществляет синтез и распознавание речи. Последнее, кстати, очень удобно. Дело в том, что Speech-функции пакета вызывают дополнительные модули, которые реализуют низкоуровневую

работу с речью. Благодаря этому разработчики других фирм могут создавать массу модулей синтеза и распознавания речи. Соблюдая правила, описанные в соответствующей документации, разработчики могут состязаться в борьбе за качество своих синтезаторов и распознающих модулей.

Каждый модуль (engine, или движок) синтеза или распознавания речи можно классифицировать по нескольким параметрам. Прежде всего они различаются по наименованию фирмы, их разработавшей. Кроме того, все эти модули имеют собственное название — ProductName, а также различного рода технические параметры. Для речевых синтезаторов это: язык, имя голоса (у голосового движка L&H — русский, Борис и Светлана, а у Digalo — русский, Николай), его пол, возраст, стиль, поддерживаемые интерфейсы и характерные особенности (features). Модули распознавания речи кроме общих параметров, имеют свои особенности, отличающиеся от свойств модулей синтеза речи. Из них можно назвать такие, как схема распознавания речи, максимальный размер словаря, поддерживаемые типы грамматик и т. д. Однако в числе общих параметров, кроме имени, фирмы-производителя и языка, есть еще один, который позволяет уникально идентифицировать голос (!) модуля в системе — это параметр `ModeID`. Как вы думаете, какой программный тип больше всего подойдет к этому параметру? Естественно, `TGUID` — тип, который генерируется случайным образом и имеет полную статистическую уникальность. По заверениям Microsoft всех возможных значений этого типа нам хватит на необозримое будущее. Через специальные интерфейсы SAPI можно определять все эти атрибуты.

Замечание

Иногда важно не путать непосредственно голосовой движок (Engine) и голос (Voice). Понятие речевого модуля может ассоциироваться с параметром `EngineID`, с которым мы познакомимся далее, — так вот, оно не меняется в различных голосовых модулях. Различные дикторы идентифицируются по `ModeID`. Именно дикторов выбирают, когда обозначают, каким голосом будет читаться текст. Так как в большинстве случаев, говоря "дикторы", "голоса", "TTS-генераторы", "синтезаторы", имеют в виду именно первых, то понятие "движок" употребляют как синоним. В тех случаях, когда это важно, мы будут разграничивать понятия голоса и голосового движка, иначе же будем считать их идентичными.

Кроме атрибутов модулей, движки могут поддерживать различные типы диалоговых окон. Для синтезаторов это диалоговые окна **About** (О модуле), **General** (Главное), **Lexicon** (Лексический словарь), **Translate** (Транскрипция). Для программ распознавания речи, кроме первых трех названных окон, существуют диалоговые окна **Training**, или **TrainGeneral** (Тренировка диктора), **TrainMic** (Корректировка чувствительности микрофона).

В настоящее время модули синтеза речи (Engine, TTS-генераторы, синтезаторы, движки) разрабатываются, насколько мне известно, следующими фирмами.

- AcuVoice ((**http://www.acuvoice.com**)
- Digalo (**http://www.digalo.com**)
- Digital Equipment Corporation (**http://www.digital.com/oem/products/dectalk/dtalk.htm**)
- Elan Informatique (**http://www.elan.fr**)
- Eloquent Technology (**http://www.eloq.com**)
- First Byte (**http://www.fbyte.com)**
- IBM (**http://www.ibm.com/viavoice**)
- Lernout & Hauspie (**http://www.lhs.com**)
- Lucent (**http://www.lucent.com/ideas2/lsap/**)
- Microsoft (**http://www.microsoft.com/iit/**)
- Telia Promotor (**http://www.promotor.telia.se/infovox/index.htm**)
- Telephonica I+D (**http://www.tid.es/english/home2.html**)
- TELES AG (**http://www.teles.de**)

Среди больших компаний можно выделить Lernout & Hauspie. В ее архиве есть синтезаторы для большинства языков, среди которых: русский, английский, французский, итальянский, испанский, немецкий, португальский, японский. Может быть, русские голоса виртуальных Бориса и Светланы и не годятся для чтения сказок на ночь, однако именно они служат неформальными лидирующими модулями синтеза русской речи в силу того, что являются бесплатными (впрочем, как и все модули этой фирмы) и более приемлемы по размеру (до 3 Мбайт).

Более качественным синтезом речи отличаются модули фирмы Digalo. Правда, у них не такой большой спектр поддерживаемых языков, хотя русский среди них есть. Впечатления от голоса Николая "от Digalo" у автора сложились, правда, несколько противоречивые: в принципе, он меньше других ошибается в ударениях, но невероятно грустная его интонация опять же будет препятствовать чтению сказок. Один из посетителей моего сайта, Genn, высказал насчет этого Николая такое мнение: "русский белоэмигрант-парижанин, князь или граф, с поставленной дикцией".

Модуль английской речи производит лучшее впечатление, кажется, что ему присущ больший акцент, чем это заметно в движках Lernout & Hauspie. Правда, в некоторых источниках можно обнаружить замечание о том, что слух англоязычного народа синтезированную родную речь воспринимает

даже хуже, чем наш — синтезиррованную русскую. В это можно поверить, т. к. сложные правила чтения английского языка описать формально, конечно, труднее.

Еще два немаловажных факта препятствуют распространению модулей синтеза речи Digalo. Во-первых, все они стоят денег, а во-вторых, "весят" порядка 10 Мбайт. Более того, новые версии движков работают только вместе с собственной программой для чтения текстов, а предыдущие версии можно скачать теперь только с сайтов третьих лиц.

Еще одна фирма — Elan (**http://www.elantts.com**) — также занимается речевыми синтезаторами, правда в их арсенале до сего момента отсутствует русский язык.

С модулями распознавания речи дела обстоят намного хуже, их разрабатывают всего 8 фирм:

- Dragon (**http://www.dragonsys.com**)
- IBM (**http://www.ibm.com/viavoice**)
- Lernout & Hauspie (**http://www.lhs.com**)
- Lucent (**http://www.lucent.com**)
- Microsoft (**http://www.microsoft.com**)
- OMRON (**http://www.omron.co.jp/home-e.html**)
- TELES AG (**http://www.teles.de**)
- Voice Control Systems (**http://www.voicecontrol.com**)

Из наиболее широко применяемых модулей распознавания речи можно отметить только движок гиганта — производителя программного обеспечения Microsoft Speech Recognition Engine v4.0 для английской речи. Существует также и модуль для японской речи, однако, по известным причинам, он нас мало интересует.

Итак, рассмотрев арсенал предлагаемых рынком программных средств, можно выделить следующий алгоритм действий. Первым делом нужно скачать все необходимые компоненты, а затем установить пакет функций Speech API, а также необходимые речевые синтезаторы и модули распознавания речи. Пакет функций Speech API самостоятельно устанавливается в папку Speech корневого каталога системы (т. е. для Windows 9*x*/XP по умолчанию Windows\Speech, для Windows 2000 — Winnt\Speech). Полный набор ссылок можно найти в *приложении 1.*

Речевые модули устанавливаются в разные каталоги и, как правило, их выбрать нельзя, но вот стандартные ключи системного реестра известны. Ко-

нечно же, это ключи в разделе `HKEY_CLASSES_ROOT\CLSID`, а также ключи в следующих, перечисленных ниже разделах.

- Для синтеза речи:
  - `HKEY_LOCAL_MACHINE\Software\Voice\TextToSpeech\Engine` или
  - `HKEY_LOCAL_MACHINE\Software\CompanyName` и `HKEY_CURRENT_USER\Software\CompanyName`.
- Для распознавания речи:
  - `HKEY_LOCAL_MACHINE\Software\Voice\SpeechRecognition\Engine` или
  - `HKEY_LOCAL_MACHINE\Software\CompanyName` и `HKEY_CURRENT_USER\Software\CompanyName`.

Почти все существующие модули распознавания речи являются дикторозависимыми, т. е. привязываются к определенному голосу. Однако майкрософтовский движок может работать и без выбора определенного диктора, соответсвенно качество распознавания от этого снижается. Практически любой модуль распознавания речи требует после его установки настройки микрофона для более качественного распознавания речи (сюда входит подстройка уровня громкости, а также выбор типа микрофона — настольный, ручной и т. п), и настройки (тренировки) движка под конкретного пользователя, т. е. на его голос.

Рассмотрим этот процесс на примере наиболее популярного движка от фирмы Microsoft. После непродолжительного копирования файлов инсталлятор предлагает настроить уровень динамиков и микрофона. Здесь вас попросят подстроить бегунок громкости колонок для комфортной работы. Затем предложат выбрать тип микрофона и прочитать фразу с экрана обычным голосом, чтобы мастер смог скорректировать уровень громкости микрофона (рис. 3.1).

Далее вас попросят провести небольшую тренировку голоса, чтобы движок смог настроиться на диктора. Движок настраивается под текущего пользователя, так что модуль можно настраивать и на других пользователей системы. Возможно, инсталлятор предложит выбрать сессию для тренировки. Это просто название текста, который вам нужно прочитать, а движок во время чтения будет привыкать к вашему голосу и манере говорить (рис. 3.2).

После того как вы прочитаете весь текст, модуль начнет собирать информацию о распознанных словах, и вам придется немного подождать. В последнем диалоговом окне выдаются результаты (рис. 3.3). Если процент распознавания невысок, то, скорее всего, модуль предложит потренироваться еще. Последнее, однако, необязательно, хотя для достижения хороших результатов желательно.

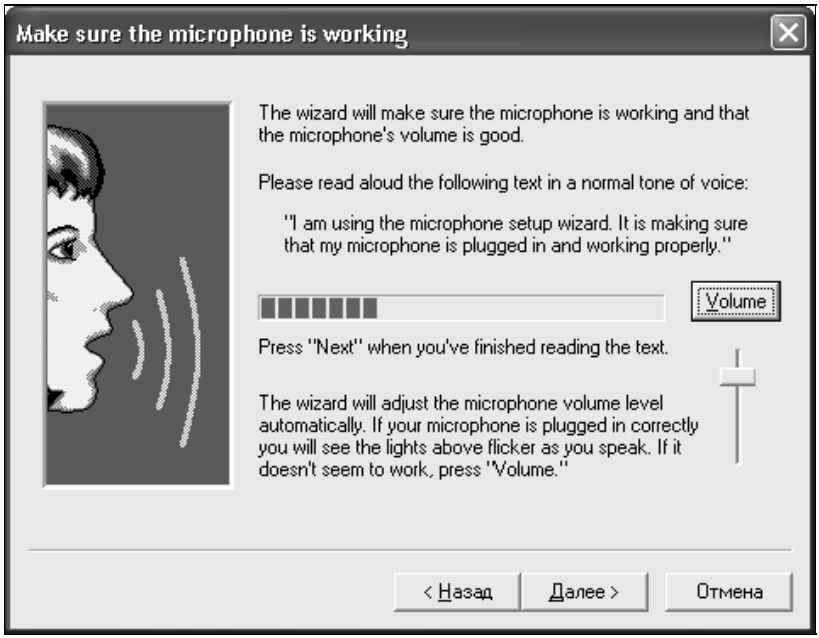


Рис. 3.1. Настройка громкости микрофона движка распознавания речи от Microsoft

Рис. 3.2. Обучение движка распознавания речи

Рис. 3.3. Окно результатов мастера тренировки голоса

Для программирования речи в Delphi нам потребуется заголовочный файл с интерфейсами, необходимыми типами и константами. Это модуль speech.pas, он находится на прилагаемом компакт-диске. Модуль работает с разными версиями Delphi и проверен в версиях 5, 6 и 7. Его попросту необходимо подключать к проектам Delphi — например, размещать в каталоге проекта. Однако более удобный способ — это создать папку по вашему усмотрению в том каталоге, где вы храните пакеты компонентов, положить в нее модуль speech.pas, в диалоговом окне **Environment Option** (Опции окружения) доступного через пункт меню **Tools | Environment Option** (Инструменты | Опции окружения), на вкладке **Library** (Библиотека) в поле ввода **Library path** (Пути к библиотекам) указать путь расположения созданного каталога. После этого можно подключить модуль через его имя, не копируя сам модуль каждый раз в каталог проекта. Модуль этот переписан с аналогичного для C++, т. е. с speech.h, и если бы его у вас не было, и я не скопировал бы его на дискету, то модуль пришлось бы скачивать из Интернета, например, со страницы автора этой книги или с домашнего сайта модуля **http://delphi-jedi.org**.

Speech-технологии

Прикладной интерфейс программирования Microsoft Speech API (SAPI) включает в себя все основные речевые технологии, поэтому еще раз обозначим их в понятиях SAPI.

- Command and Control
- Dictation
- Text to speech
- Telephony

Рассмотрим назначение этих технологий.

- Технология Command & Control — предусматривает распознавание речи в виде отдельных слов или фраз из списка ожидаемых компьютером. Подобная функция может использоваться для управления голосом. Например, вместо того чтобы искать пункт меню для запуска печати мышью, вы можете отдать команду голосом: "Print current page". Команды могут содержать как статическую часть, так и вариантную, например: "Позвонить *{номер}*" или "Разбуди меня в *{время}*". Это маленькое расширение дает богатые возможности для применения, а также сокращает количество включений однотипных команд.
- Технология Dictation — реализует ввод распознавания речи произвольного содержания, т. е. предназначена для диктовки текста. Ее можно использовать, например, для заполнения простейших анкетных форм, а также для введения текста в документы соответствующих текстовых редакторов.
- Технология Text to speech — позволяет синтезировать связную речь из передаваемого текста.
- Технология Telephony — представляет собой смесь распознавания и синтеза речи. Типичные функции для VoiceTelephony — это расширенный автоответчик, голосовая почта, голосовая справочная служба и т. п. Используя компьютер, обычный модем и телефон, можно пользоваться удаленным голосовым администрированием своего ПК. Например, вы забыли установить таймер автоматической записи на свой видеомагнитофон, и поэтому решили попросить компьютер записать вашу любимую передачу через его карту TV-тюнер. Вы звоните себе домой, трубку снимает ваш модем, после чего происходит следующий диалог:

  Компьютер: "Привет! Кто это?"

  Вы: "Привет! Это хозяин!"

  Компьютер: "Скажите ваш пароль для управления".

  Вы: "Такой-то пароль".

Компьютер: "Вы вошли в систему! Что вы хотели бы сделать?"

Вы: "Запиши передачу!"

Компьютер: "Записать сегодня передачу по TV?"

Вы: "Да".

Компьютер: "Сообщите канал и время!"

Вы: "Первый канал, в пятнадцать часов тридцать минут".

Компьютер: "Первый канал, в 15:30. Верно?"

Вы: "Да".

Выглядит заманчиво, не правда ли? Однако подобные игрушки требуют наличия голосовых модемов, причем желательно дуплексных. Это означает, что модем должен быть способен одновременно воспроизводить и записывать звуки. Иначе вам вряд ли понравится, если невозможно будет прервать речь компьютера и дать свою команду в процессе долгого перечисления всех доступных команд. С полудуплексными модемами также можно работать, однако в этом случае приложения должны предусматривать такую возможность в коде.

Компоненты SAPI

При создании любая система раскладывается на несколько основных компонентов (составных частей), и таким образом человеку удобнее воспринимать ее. В прошлой главе мы знакомились со структурой модели Agent Control, объекты которой располагаются в иерархическом порядке. Порядок объектов в SAPI не вполне иерархический, т. е. он не представляется в виде дерева. Объекты располагаются там в несколько уровней (рис. 3.4).

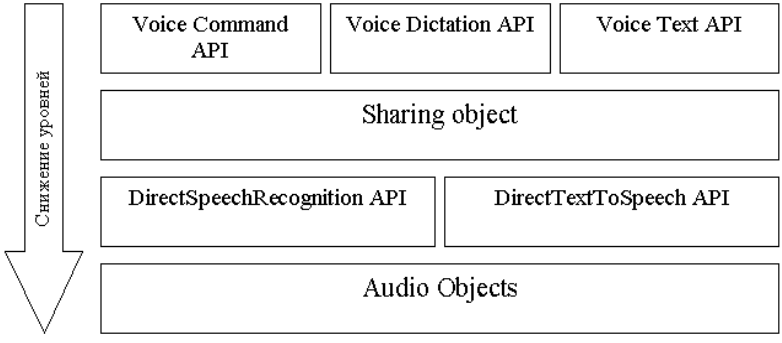


Рис. 3.4. Схема расположения различных компонентов SAPI

На самом верхнем уровне находятся такие интерфейсы, как Voice Commnad API, Voice Dictation API, Voice Text API. Это обосновано тем, что данные компоненты предоставляют несколько ограниченный уровень доступа к речевым модулям. Зато интерфейсы управления у них намного проще, чем у следующих компонентов.

В противоположность им существуют прикладные интерфейсы Direct Speech Recognition API и Direct Text To Speech API, они считаются низкоуровневыми (возможно, и более быстрыми), т. к. предоставляют самый широкий доступ к управлению модулями распознавания и синтеза речи соответственно. Ниже этих компонентов — только прямое обращение к аудиообъектам (Audio Objects), которые обеспечивают доступ к звуковым устройствам.

Возможно, у вас возникнет законный вопрос: "А где здесь VoiceTelephony?" Действительно, в схеме, имеющейся в документации к Microsoft Speech API, не указано место для Voice Telephony API, однако в Speech API SDK этот интерфейс имеет статус высокого. Голосовая телефония предоставляет доступ к синтезу речи, распознаванию речи, а также к волновому синтезу на простых и многоканальных голосовых телефонных устройствах.

Как и в случае с Microsoft Agent, в Speech API под указанными компонентами (как составными частями SAPI) понимаются COM-объекты с определенными интерфейсами. Рассмотрим теперь каждый компонент более подробно.

Voice Command API — прикладной интерфейс высокого уровня, обеспечивающий распознавание голосовых команд. Он предназначен специально для создания приложений, поддерживающих речевое управление, т. е. голосовые команды. Этот API-компонент настроен на распознавание специально подготовленных фраз и выражений. Работа с Voice Command API похожа на создание виртуального меню, куда добавляются команды для распознавания речи. Большинству приложений достаточно иметь одно такое меню в главном окне для необходимых команд и по одному — на каждое диалоговое окно, чтобы управлять таким командами, как "Yes", "No" и т. д. Если речевой модуль успешно распознает волновые данные, поданные на вход звукового устройства, то соответствующий COM-объект проверяет наличие распознанной фразы в списке команд и при успешном результате информирует об этом главное приложение.

Другой высокоуровневый компонент — Voice Dictation API — позволяет осуществлять диктовку в приложениях и таким образом заполнять формы и текстовые поля ввода. Компонент построен интересным образом: здесь, так же как и в Voice Command API, все операции производятся как будто с невидимым полем редактирования, при этом в случае его изменения VoiceDictation уведомляет об этом приложение, чтобы оно могло предоставить пользователю адекватную информацию.

Прикладной интерфейс программирования Voice Text API работает по принципу конвертера, переводящего напечатанные слова в связную речь. На входной буфер подается текст, а соответствующие COM-объекты передают этот текст на синтез речевому модулю, причем главное приложение извещается о начале чтения, о его завершении, а также о текущей воспроизводимой фонеме.

Функционирование интерфейса программирования Voice Telephony API (в просторечии "телефония") несколько непривычно и принципиально отличается от работы с такими обычными средствами управления, как кнопки и текстовые поля редактирования, которые графически отображаются на экране. Обычно приложение не в силах контролировать переходы между элементами управления, т. е. оно представляет ряд таких элементов, однако программным способом невозможно предположить, каким именно из них пользователь воспользуется в определенный момент. Телефония чем-то напоминает DOS-консоль, подобную командной строке, правда, если инструмент не был предложен приложением, пользователь не может его выбрать. Ему позволено лишь отвечать на вопросы и выводить дополнительную информацию. Так, например, типичные вопросы телефонии предполагают односложные (замкнутые) ответы типа "Да/Нет/Назад", а также ввод цифр (номеров) времени. Тем не менее, голосовая телефония все-таки предусматривает анализ свободных вопросов, но эта функция также должна быть предложена приложением пользователю.

Под "мифическими" компонентами Sharing Object подразумеваются объекты, которые позволяют интерфейсам более высокого уровня совместно использовать речевые модули.

Компонент DirectSpeechRecognitionAPI — это низкоуровневый интерфейс прямого распознавания речи. Работа с ним предполагает непосредственное общение с речевыми модулями. Конечно, оно происходит не напрямую с исполняемыми файлами, а через специально созданные интерфейсы. Если в случае с VoiceCommand и VoiceDictation достаточно программно задать небольшое количество команд для запуска этих функций (типа инициализации, выбора нужного речевого модуля и активации распознавания), то при использовании DirectSpeechRecognition API все намного сложнее.

Кратко описать алгоритм работы с прикладным интерфейсом программирования DirectSpeechRecognition API можно следующим образом. Первоначально настраивается связь с необходимым аудиоустройством и специальным интерфейсом компонента Audio Objects. Им может быть как вход нужной звуковой карты (т. е. распознавание с микрофона), так и волновой файл. Второй шаг включает в себя получение указателей на другие необходимые интерфейсы COM-объектов, например, для связи и поиска модулей распознавания речи, а также интерфейсы грамматики и атрибутов распознавания речи. На следующем шаге осуществляется поиск установленных рече-

вых модулей в системе и отбор их по необходимым критериям (таким как язык, диалект, фирма-производитель и др). После этого выбирается нужный модуль и через центральный интерфейс указывается, что именно выбранный модуль будет распознавать речь. На предпоследнем шаге для выбранного модуля загружаются определенные грамматические правила, т. е. набор слов, фраз, шаблонов, которые будут распознаваться двигателем. Грамматика может быть контекстно-свободной (context-Free), ограниченной доменом (limited-domain) и грамматикой диктовки (dictation). На этом же шаге выполняется регистрация специального объекта, реализующего необходимый интерфейс notification sink, через который приложение будет уведомлять о происходящих событиях. В завершение остается только активизировать функцию распознавания речи.

Применение интерфейса Direct Text To Speech API также более громоздко, чем применение Voice Text API, но зато на этом уровне разработчику предоставляется значительно больший круг инструментов по управлению модулями синтеза речи.

Алгоритм работы с прямым синтезом речи чем-то похож на алгоритм для DirectSpeechRecognition API. Тут на первом шаге также нужно организовать связь с аудиоустройством или звуковым файлом, в который будет производиться запись аудиопотока. Затем также следует подобрать нужный речевой синтезатор, соответствующий нужному языку. Затем настроить необходимые параметры синтеза речи и, по необходимости, зарегистрировать объект, принимающий все события (иногда в этом нет надобности). Наконец, в нужный момент следует передать на чтение текст из буфера.

Стоит отметить, что, кроме использования интерфейса компонентов напрямую, можно работать с внедренными элементами ActiveX так, как мы это делали в случае с MS Agent. В *гл. 2* мы преимущественно работали с визуальным компонентом Agent для связи с управляющим элементом Agent Control, но для речевых технологий все будет иначе. Большей частью мы будем обращаться напрямую к интерфейсам различных компонентов SAPI, т. к. это полезно для понимания работы SAPI. Если вы разберетесь с функционированием компонентов на уровне интерфейсов, то понять, как применять соответствующий элемент ActiveX, не составит для вас труда.

Сначала мы познакомимся с интерфейсами синтеза речи, затем с интерфейсами распознавания речи. В каждой логической части будем сначала говорить о высокоуровневом интерфейсе, а затем о низкоуровневом. Объекты Shared Object и Audio Object мы будем рассматривать по мере их использования. Кроме того, необходимо сказать, что в данной книге не будет рассматриваться компонент Voice Telephony API.

3.3. Интерфейс синтеза речи Voice Text API

В этом разделе мы рассмотрим интерфейс синтеза речи Voice Text API. Как уже отмечалось, это компонент более высокого уровня — следовательно, его применение проще, чем применение интерфейса DirectTextToSpeech API, однако и менее гибко. В этом разделе мы напишем также простейшую программу синтеза речи на основе Voice Text API.

Необходимые интерфейсы и объекты

Любая программа, работающая с COM-объектами, начинает работу с получения указателей на интерфейсы основного объекта приложения. Так, основным объектом для Voice Text API является объект `VoiceText` с его главным интерфейсом `IVoiceText`. Все основные манипуляции начинаются именно с него. Последовательность действий можно описать следующим образом:

1. Получение указателя на `IvoiceText`.
2. Получение указателя на `IVTxtAttributes`.
3. Создание и регистрация объекта, реализующего интерфейс `IVTxtNotifySink`.

Через интерфейс `IVoiceText` чтение запускается, приостанавливается (пауза) и прекращается, через него же регистрируется объект, получающий уведомления (notification sink) (см. табл. 3.1). Интерфейс `IVTxtAttributes` предназначен для изменения атрибутов синтеза — таких как речевой модуль, скорость чтения, а также устройство вывода звука.

Методы интерфейса `IVoiceText` приведены в табл. 3.1.

Таблица 3.1. *Методы интерфейса* `IVoiceText`

Метод	Описание
`function Register(pszSite, pszApplication: PAnsiChar; NotifyInterface: IVTxtNotifySinkA; IIDNotifyInterface: TGUID; dwFlags: DWORD; pSiteInfo: PVTSiteInfo): HResult; stdcall;`	Метод регистрирует объект, реализующий интерфейс notification sink для получения сообщений, указанных во флаге `dwFlags`. Первый, строковый параметр отражает место назначения вывода звука. Он может быть равен "LineX", где X — номер телефонной линии. Если он равен `nil`, то это звуковое устройство компьютера. Второй параметр идентифицирует приложение, этот параметр не может быть равен `nil` и должен содержать произвольную строку, хотя бы "MyApp".

Таблица 3.1 *(продолжение)*

Метод	Описание

	Последний параметр определяет указатель на структуру `TVTSiteInfo`, которая хранит начальные установки атрибутов чтения, такие как идентификатор голоса, скорость и идентификатор звукового устройства вывода компьютера. Если он равен `nil`, то настройки по умолчанию загружаются из системного реестра.
	В случае ошибки метод возвращает одно из следующих значений (здесь и далее описание кодов ошибок см. в табл. 3.3):
	`VTXTERR_INVALIDPARAM`
	`VTXTERR_OUTOFMEM`

`function Speak(pszSpeak: PAnsiChar; dwFlags: DWORD; pszTags: PAnsiChar): HResult; stdcall;`	Метод запускает синтез речи для текста из первого параметра. Вторым параметром указывается тип текста и его приоритет *(см. далее)*.
	В последнем параметре указываются необходимые речевые теги, которые должны быть применены к читаемому тексту *(см. разд. 3.4, подразд. "Теги")*.
	В случае ошибки метод возвращает одно из следующих значений:
	`VTXTERR_INVALIDMODE`
	`VTXTERR_INVALIDPARAM`
	`VTXTERR_NOTENABLED`
	`VTXTERR_OUTOFMEM`
	`VTXTERR_QUEUEFULL`
	`VTXTERR_WAVEDEVICEBUSY`

`function StopSpeaking: HResult; stdcall;`	Метод останавливает чтение текущего фрагмента текста и очищает очередь воспроизведения.
	Возвращаемые коды ошибок:
	`VTXTERR_INVALIDMODE`
	`VTXTERR_NOTENABLED`
	`VTXTERR_OUTOFMEM`

`function AudioFastForward: HResult; stdcall;`	При чтении фрагмента текста метод перемещает виртуальный курсор чтения примерно на следующее предложение.

Таблица 3.1 *(продолжение)*

Метод	Описание

	Метод эффективен при больших фрагментах и длинных предложениях. Однако курсор не всегда перемещается точно на следующее предложение.
	
	Если в очереди остается фрагмент текста, меньший чем предложение, метод перемещает курсор чтения в конец очереди и останавливает синтез. Если никакой текст не воспроизводится, то метод ничего не делает и не возвращает никакой ошибки.
	
	Коды ошибок:
	`VTXTERR_INVALIDMODE`
	`VTXTERR_INVALIDPARAM`
	`VTXTERR_NOTENABLED`
	`VTXTERR_NOTENOUGHDATA`
	`VTXTERR_OUTOFMEM`

`function AudioPause: HResult; stdcall;`	Метод приостанавливает чтение. Типичная пауза.
	
	Коды ошибок:
	`VTXTERR_INVALIDMODE`
	`VTXTERR_NOTENABLED`
	`VTXTERR_OUTOFMEM`

`function AudioResume: HResult; stdcall;`	Метод восстанавливает синтез после применения метода `AudioPause`.
	
	Коды ошибок:
	`VTXTERR_INVALIDMODE`
	`VTXTERR_NOTENABLED`
	`VTXTERR_OUTOFMEM`

`function AudioRewind: HResult; stdcall;`	Метод как бы "перематывает" воспроизведение текста примерно на одно предложение назад. Действие метода противоположно действию метода `AudioFastForward`. Если метод вызывается по достижении конца очереди, то возобновляется чтение примерно последнего предложения в потоке. Если метод запускается, когда никакой текст не воспроизводится, то он ничего не делает и не возвращает никакой ошибки.

Таблица 3.1 *(окончание)*

Метод	Описание

	Коды ошибок:
	`VTXTERR_INVALIDMODE`
	`VTXTERR_NOTENABLED`
	`VTXTERR_NOTENOUGHDATA`
	`VTXTERR_OUTOFMEM`

Через интерфейс `IVTxtNotifySink` объект `VoiceText` уведомляет главное приложение обо всех происходящих событиях, поэтому если разработчик создает объект, реализующий этот интерфейс, то в главном приложении можно будет описывать реакцию на эти события в виде соответствующих методов. Методы интерфейса уведомлений `IVTxtNotifySink` приведены в табл. 3.2.

Таблица 3.2. *Уведомления (notification sink), обрабатываемые интерфейсом* `IVTxtNotifySink`

Notifies	Описание

`function AttribChanged(dwAttribute: DWORD): HResult; stdcall;`	Уведомляет об изменении атрибутов. Параметр может иметь одно из следующих значений:
	`IVTNSAC_DEVICE`
	`IVTNSAC_ENABLED`
	`IVTNSAC_SPEED`
	`IVTNSAC_TTSMODE`
	`IVTNSAC_VOLUME`

`function Visual(cIPAPhoneme: WideChar; cEnginePhoneme: AnsiChar; dwHints: DWORD; pTTSMouth: PTTSMOUTH): HResult; stdcall;`	Извещает о текущей читаемой фонеме. Первый параметр — это фонема в интернациональном фонетическом алфавите (International Phonetic Alphabet, IPA). Второй — фонема, указываемая речевым синтезатором. Третий параметр — это тип выражения. Он может принимать следующие значения: `TTSNSHINT_COMMAND` — команда; `TTSNSHINT_EMPHASIS` — часть подчеркиваемого слова; `TTSNSHINT_EXCLAMATION` — часть восклицания; `TTSNSHINT_QUESTION` — часть вопроса; `TTSNSHINT_STATEMENT` — часть инструкции.

Таблица 3.2 *(окончание)*

Notifies	Описание
	Последний параметр является адресом структуры `TTSMOUTH`, которая содержит информацию о положении рта и губ человека, произносящего данную фонему (см. описание в модуле speech.pas на прилагаемом компакт-диске)
`function Speak(pszText: PAnsiChar; pszApplication: PAnsiChar; dwType: DWORD): HResult; stdcall;`	Уведомляет приложение, когда в очередь чтения добавляется фрагмент текста. Первый параметр — собственно текст, второй — идентификатор приложения, которое добавило этот текст, последний параметр содержит то же значение, что и параметр `dwFlags` в методе `IVoiceText.Speak` (см. табл. 3.1)
`function SpeakingStarted: HResult; stdcall;`	Извещает о начале чтения очереди текста
`function SpeakingDone: HResult; stdcall;`	Уведомляет о завершении воспроизведения очереди текста

Из дополнительных интерфейсов можно выделить интерфейс `IVTxtDialogs`, с помощью которого вызываются различные диалоги речевых модулей, например, диалоговые окна **About** (О модуле), **General** (Главное), **Lexicon** (Словарь), **Translate** (Перевод).

Описание кодов ошибок, возникающих при обращении к тем или иным методам, приведены в табл. 3.3.

Таблица 3.3. *Коды ошибок для методов объекта* `VoiceText`

Константа	Значение	Описание
`VTXTERR_INVALIDMODE`	$80040206	Неверный `TTSMode` или приложение (движок) не зарегистрировано для использования `VoiceText`
`VTXTERR_INVALIDPARAM`	$80070057	Неверный параметр
`VTXTERR_INVALIDWINDOW`	$8004000F	Неверное окно
`VTXTERR_NOTENABLED`	$80040719	`VoiceText` недоступен
`VTXTERR_NOTENOUGHDATA`	$80040201	Недостаточно данных (текстовый поток был передан без начала или без окончания)
`VTXTERR_NOTSUPPORTED`	$80004001	Диалоговое окно не поддерживается движком

Таблица 3.3 *(окончание)*

Константа	Значение	Описание
VTXTERR_OUTOFMEM	$8007000E	Не хватает памяти
VTXTERR_QUEUEFULL	$8004020A	Очередь воспроизведения заполнена
VTXTERR_WAVEDEVICEBUSY	$80040203	Звуковое устройство занято

Итак, мы разобрали основные необходимые нам интерфейсы и теперь можем приступить к программированию.

Первая программа чтения текстов

Хватит теории! Давайте напишем несложную программу чтения текстов. Создадим пустую заготовку приложения и на главной форме разместим текстовое поле `TEdit` и кнопку `TButton`, по нажатию которой будет происходить чтение текста.

Во время запуска приложения нужно получить указатели на все необходимые интерфейсы, а также проделать другие действия по инициализации. Добавим нужные переменные интерфейсного типа в раздел `public` нашей формы. Обычно поступать так удобнее, нежели объявлять переменные в разделе `var` модуля. Хотя это дело вкуса, и, возможно, ваше приложение может потребовать объявления переменных в другом месте. Описание класса формы выглядит следующим образом:

```
TForm1 = class(TForm)
  Edit1: TEdit;
  Button1: TButton;
public
  { Public declarations }
  gpIVTxt: IVoiceText;
end;
```

Чтобы для описания был доступен интерфейс `IVoiceText`, не забудьте подключить в разделе `uses` модуль speech.pas.

Теперь в обработчик события создания главной формы приложения запишем следующий код:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 CoCreateInstance(CLSID_VTxt, nil, CLSCTX_ALL, IID_IVoiceText, gpIVTxt);
 OleCheck(gpIVTxt.Register(nil, 'MyApp', nil, GUID_NULL, 0, nil));
end;
```

Первой строкой, как можно догадаться, мы обращаемся к библиотеке COM за указателем на интерфейс `IVoiceText`. Затем нам нужно зарегистрировать приложение для использования в нем объекта `VoiceText`. Это делает метод `Register` интерфейса `IVoiceText`. Данный метод также регистрирует класс для обработки уведомлений (notification sink), т. к. до тех пор пока мы не пользуемся уведомлениями, третий и четвертый его параметры имеют нулевые значения. Константа `GUID_NULL` отражает нулевое значение типа `TGUID`. Кстати, она объявлена в модуле `ActiveX`, поэтому его нужно подключить в разделе `uses`. Единственный обязательный параметр в этой процедуре — это идентификатор приложения (он не может быть равен `nil`).

Процедура `OleCheck`, из которой запускается метод `Register`, нужна для того, чтобы генерировать исключения `EOleSysError` в случае ошибки. Иначе приложение будет умалчивать о случаях, когда функция, обращающаяся с COM-объектом, имеет код ошибки. Благодаря этой процедуре вы всегда сможете отловить точное место ошибки в приложении, использующем объекты COM и OLE. Если возвращаемое методом `Register` значение будет меньше нуля (что несет информацию о произошедшей ошибке), то процедура `OleCheck` генерирует исключение и вы увидите соответствующее сообщение. Данная процедура объявлена в модуле `ComObj`, поэтому не забудьте подключить его в разделе `uses`.

Теперь осталось запустить в нужное время синтез речи. Как мы уже договорились, происходить это будет при щелчке на кнопке `TButton`, поэтому в обработчик события щелчка мыши нужно вставить следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 gpIVTxt.Speak(PChar(Edit1.Text), VTXTST_STATEMENT or VTXTSP_NORMAL, nil)
end;
```

Первым параметром передается необходимый текст, вторым мы указываем тип выражения и его приоритет. В данном случае это обычное выражение с нормальным приоритетом.

Так как воспроизведение текста — это операция асинхронная (так же, как и проигрывание анимации MS Agent), то каждый запуск метода `Speak` ставит фрагмент текста в очередь на воспроизведение. Для каждого такого фрагмента могут быть заданы через соответствующий параметр тип текста и тип приоритета. Тип текста определяет интонацию, с которой будет произноситься текст, а приоритет — порядок вставки его в общую очередь. В табл. 3.4 указаны возможные значения параметра, задающего тип текста для флага `dwFlags` в методе `Speak` интерфейса `IVoiceText`.

Таблица 3.4. *Значения констант, задающих тип текста для флага* `dwFlags`

Константа	Значение	Описание
VTXTST_STATEMENT	$00000001	Обычный текст (значение по умолчанию)
VTXTST_QUESTION	$00000002	Текст, представляющий собой вопрос
VTXTST_COMMAND	$00000004	Команда, инструкция пользователю
VTXTST_WARNING	$00000008	Предупреждение
VTXTST_READING	$00000010	Обычный текст из документа
VTXTST_NUMBERS	$00000020	Текст, состоящий из цифр
VTXTST_SPREADSHEET	$00000040	Текст из электронной таблицы, такой как колонка цифр

Обычно влияние параметра типа текста невозможно уловить на слух, и некоторые синтезаторы его не воспринимают, а ориентируются только на знаки препинания в реальном тексте и на его содержание. Поэтому его можно всегда оставлять равным значению по умолчанию. А вот параметр приоритета может весьма пригодиться в случае, если какой-либо текст нуждается в ином приоритете (например, если его нужно прочитать немедленно). Все значения параметра приведены в табл. 3.5.

Таблица 3.5. *Значения констант, задающих тип приоритета для флага* `dwFlags`

Константа	Значение	Описание
VTXTSP_VERYHIGH	$00000080	Самый высокий приоритет. При этом значении текущий читаемый текст, если он имеет более низкий приоритет, прерывается и воспроизводится заданный текст. После завершения воспроизведения продолжает читаться прерванный текст
VTXTSP_HIGH	$00000100	При этом значении приоритета текст ставится в очередь сразу после текущего фрагмента
VTXTSP_NORMAL	$00000200	Обычный приоритет, значение по умолчанию. В этом случае блок текста вставляется в конец очереди

Данный пример реализован в проекте 01_VoiceText. Попробуйте откомпилировать программу. Введите в поле редактирования произвольный текст и запустите чтение. Вполне возможно, что программа прочитает его не на том языке, либо не с таким акцентом (например, английский текст русские движки (т. е. синтезаторы) L&H читают с "рязанским" акцентом), либо во-

обще откажется читать. Это зависит от того, какой речевой синтезатор установлен по умолчанию. Так как мы не указывали какой-либо специальный модуль синтеза речи, то настройка прочитана из реестра. Соответственно, язык текста может не совпадать с языком модуля. Обещаю, что в конце концов мы избавимся от этих ошибок, и тексты будут читаться на любых языках.

Совершенствуем первую "читалку"

Продолжим совершенствовать нашу программу. Добавим ей функции паузы и остановки чтения, а также функции получения уведомлений.

Для обработки событий, а вернее для получения так называемых уведомлений (notification sink), необходимо описать класс, реализующий интерфейс `IVTxtNotifySink`, и объект этого класса. Как мы уже знаем из *гл. 1*, самым удобным предком для подобных объектов является класс `TInterfacedObject`, т. к. в нем объявлены методы `QueryInterface`, `_AddRef` и `_Release`. Таким образом, раздел `type` модуля главной формы следует оформить так:

```
type
  TVTxtNotifySink = class;

  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
  public
    { Public declarations }
    gpIVTxt: IVoiceText;
    gpVTxtNotifySink: TVTxtNotifySink;
  end;

  TVTxtNotifySink = class(TInterfacedObject, IVTxtNotifySink)
    function AttribChanged(dwAttribute: DWORD): HResult; stdcall;
    function Visual(cIPAPhoneme: WideChar; cEnginePhoneme: AnsiChar;
      dwHints: DWORD; pTTSMouth: PTTSMOUTH): HResult; stdcall;
    function Speak(pszText: PAnsiChar; pszApplication: PAnsiChar; dwType:
DWORD): HResult; stdcall;
    function SpeakingStarted: HResult; stdcall;
    function SpeakingDone: HResult; stdcall;
  end;
```

В раздел `implementation` нужно добавить реализацию класса, отвечающую вашим потребностям. К примеру, в нашем демонстрационном проекте ука-

жем требование о том, чтобы в компоненте `TListBox` в виде лога событий отображалась информация об уведомлениях, как показано в листинге 3.1.

Листинг 3.1. Фрагмент секции `implementation` с реализацией класса `TVTxtNotifySink` (проект 01_VoiceText)

```
function TVTxtNotifySink.AttribChanged(dwAttribute: DWORD): HResult;
stdcall;
begin
 Form1.ListBox1.Items.Add('<Изменились атрибуты>');
 case dwAttribute of
  IVTNSAC_DEVICE: Form1.ListBox1.Items.Add('  Устройство вывода');
  IVTNSAC_ENABLED: Form1.ListBox1.Items.Add('  Доступность');
  IVTNSAC_SPEED: Form1.ListBox1.Items.Add('  Скорость');
  IVTNSAC_TTSMODE: Form1.ListBox1.Items.Add('  TTSMode');
  IVTNSAC_VOLUME: Form1.ListBox1.Items.Add('  Громкость');
 end;
 Result:=S_OK;
end;

function TVTxtNotifySink.Visual(cIPAPhoneme: WideChar; cEnginePhoneme:
    AnsiChar; dwHints: DWORD; pTTSMouth: PTTSMOUTH): HResult; stdcall;
begin
 Form1.ListBox1.Items.Add('Событие Visual. cIPAPhoneme:'+cIPAPhoneme);
 Form1.ListBox1.Items.Add('   cEnginePhoneme:'+cEnginePhoneme);
 Form1.ListBox1.Items.Add('   dwHints:'+IntToStr(dwHints));
 Result:=S_OK;
end;

function TVTxtNotifySink.Speak(pszText: PAnsiChar;
      pszApplication: PAnsiChar; dwType: DWORD): HResult; stdcall;
begin
 Form1.ListBox1.Items.Add('Событие Speak. pszText:'+pszText);
 Form1.ListBox1.Items.Add('   pszApplication:'+pszApplication);
 Form1.ListBox1.Items.Add('   dwType:'+IntToStr(dwType));
 Result:=S_OK;
end;
```

```
function TVTxtNotifySink.SpeakingStarted: HResult; stdcall;
begin
 Form1.ListBox1.Items.Add('---- Начал читать -----');
 Result:=S_OK;
end;

function TVTxtNotifySink.SpeakingDone: HResult; stdcall;
begin
 Form1.ListBox1.Items.Add('---- Закончил читать -----');
 Result:=S_OK;
end;
```

Подготовительные действия завершены. Теперь в обработчике события создания главной формы приложения запишем следующий код:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 CoCreateInstance(CLSID_VTxt, nil, CLSCTX_ALL, IID_IVoiceText, gpIVTxt);

 gpVTxtNotifySink:=TVTxtNotifySink.Create;
 gpIVTxt.Register(nil, 'MyApp', gpVTxtNotifySink, IID_IVTxtNotifySink,
      VTXTF_ALLMESSAGES, nil);
end;
```

Указатель на главный интерфейс `IVoiceText` объекта `VoiceText` мы получаем через обращение к библиотеке COM с помощью функции `CoCreateInstance`, т. к. мы не имеем указателя ни на какой другой интерфейс этого объекта. Остальные указатели на интерфейсы `VoiceText` можно будет получить более быстрым способом — через метод `QueryInterface` интерфейса `IVoiceText`. Так получают, например, указатели на различные дополнительные интерфейсы объекта `VoiceText` — `IVTxtAttributes`, `IVTxtDialogs` и т. д.

Затем создадим экземпляр класса `TVTxtNotifySink`, который реализует интерфейс `IVTxtNotifySink` для уведомления о событиях. Наконец, последней строкой зарегистрируем этот объект как уведомление (notification sink), сообщая, через первые два параметра, что вывод будет осуществляться через аудиоустройство компьютера (а не в телефонную линию). В качестве идентификатора приложения указываем значение "MyApp". Последние два параметра говорят о том, что мы желаем получать все возможные сообщения, а также о том, что никаких особенных настроек синтеза речи мы не используем, а значит, все параметры будут инициализироваться значениями по умолчанию из системного реестра.

Теперь, когда мы запустим синтез речи, в логе событий будет отображаться информация о происходящих событиях (рис. 3.5).

Пауза и преждевременная остановка речи осуществляются методами `AudioPause` и `StopSpeaking` интерфейса `IVoiceText`, а продолжение чтения после паузы — методом `AudioResume` того же интерфейса. Поэтому для обработки события щелчка мышью следует прописать в код необходимые действия. Здесь придется ввести переменную-флажок для определения состояния паузы (т. е. пауза сейчас или нет), чтобы при щелчке на кнопке **Читать** текст снова не загружался в очередь. За подробностями отсылаю вас к коду проекта 02_VTxt_Notify.

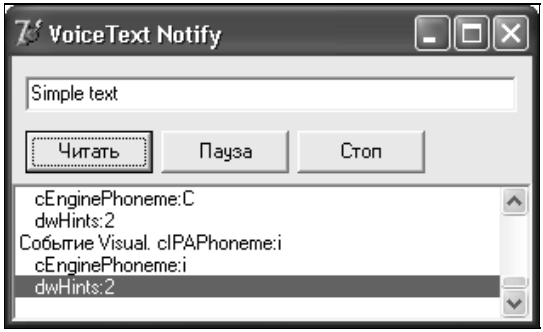


Рис. 3.5. Рабочее окно первой программы чтения голосом на Voice Text API

Изменение атрибутов синтеза речи

Менять такие параметры синтеза речи, как идентификатор речевого модуля (т. е. TTS-генератор, который синтезирует речь), скорость речи и устройство вывода, можно во время инициализации чтения. При регистрации объекта, получающего уведомления, в предыдущем проекте (02_VTxt_Notify) последний параметр был равен `nil`:

```
gpIVTxt.Register(nil, 'MyApp', gpVTxtNotifySink, IID_IVTxtNotifySink,
       VTXTF_ALLMESSAGES, nil);
```

Мы знаем, что это означает загрузку всех настроек по умолчанию из системного реестра. Оттуда берутся данные об идентификаторе синтезатора (ModeID), скорости речи и звуковом устройстве вывода по умолчанию. Применять параметры синтеза на этом этапе рационально, если используется загрузка значений из ini-файла. Например, если в предыдущем запуске было выбрано устройство, определенный движок, скорость, а в следующий раз необходимо восстановить прежние настройки. Хотя это и не всегда оп-

равданно, т. к. измененные настройки автоматически сохраняются в реестре. Однако если все-таки необходимо инициализировать атрибуты, то надо создать экземпляр структуры `TVTSiteInfo`, заполнить ее и тогда вместо значения параметра `nil` передавать указатель на переменную.

```
PVTSiteInfo = ^TVTSiteInfo;
{$EXTERNALSYM VTSITEINFO}
VTSITEINFO = record
  dwDevice: DWORD;
  dwEnable: DWORD;
  dwSpeed : DWORD;
  gModeID : TGUID;
end;
TVTSiteInfo = VTSITEINFO;
```

Помимо доступа через эту структуру и метод `Register` интерфейса `IVoiceText`, к параметрам, или атрибутам, речи существует доступ через интерфейс `IVTxtAttributes`. В предыдущем примере мы его не использовали, но с этого момента будем читать и изменять настройки атрибутов речи именно через него. Набор методов, присутствующих в интерфейсе `IVTxtAttributes`, приведен в табл. 3.6.

Таблица 3.6. *Методы интерфейса* `IVTxtAttributes`

Метод	Описание
`function DeviceGet(var dwDeviceID: DWORD): HResult`	Возвращает идентификатор аудиоустройства, через которое осуществляется вывод
`function DeviceSet(dwDeviceID: DWORD): HResult`	Меняет устройство вывода для объекта `VoiceText` по его идентификатору. Следует использовать в случае вывода не на устройство, установленное по умолчанию — например, если у вас несколько звуковых карт, или существуют другие устройства, через которые необходимо выводить звук
`function EnabledGet (var dwEnabled: DWORD): HResult`	Через этот метод можно получить информацию о том, доступен ли синтез речи в приложениях. Если возвращается значение `True`, то синтез доступен
`function EnabledSet(dwEnabled: DWORD): HResult`	Через данный метод можно изменить доступность вывода речи в приложениях. Эта настройка сохраняется автоматически при ее изменении
`function IsSpeaking (var fSpeaking: BOOL): HResult`	Возвращает значение, по которому можно определить, читается в данный момент текст или нет. Если значение параметра `fSpeaking` равно `True`, то какой-то текст читается

Таблица 3.6 *(окончание)*

Метод	Описание
`function SpeedGet(var dwSpeed: DWORD): HResult`	Возвращает скорость речи (количество слов в минуту)
`function SpeedSet(dwSpeed: DWORD): HResult`	Изменяет скорость речи
`function TTSModeGet(var gVoice: TGUID): HResult`	Возвращает идентификатор текущего TTS-режима, т. е. идентификатор "голоса" движка, через который производится чтение
`function TTSModeSet(gVoice: TGUID): HResult`	Изменяет `TTSMode`, т. е. речевой движок (синтезатор), через который производится синтез речи

Должен еще раз напомнить, что все параметры, изменяемые методами интерфейса `IVTxtAttributes`, автоматически сохраняются в системном реестре и нет необходимости при загрузке приложения вручную восстанавливать настройки. В основном, ими приходится пользоваться для текущего изменения атрибутов речи в приложении. Таким образом, мы в нашем приложении создадим интерфейс для выбора речевого синтезатора, а также для изменения скорости речи. Для простоты будем выбирать движки из раскрывающегося списка (пусть это будет элемент `ComboBox1`), содержащего предопределенный список модулей, а значение скорости вводить цифрами в поле редактирования (элемент `edSpeed`). Как вы понимаете, это не самый лучший способ, однако он вполне годится для учебного примера. Далее мы рассмотрим порядок поиска установленных в системе движков и способы выяснения возможностей модулей для определения границ диапазона скорости речи, а также других параметров. Я принципиально не хочу сейчас рассматривать вопрос о том, как осуществить поиск всех модулей, применяя интерфейс Voice Text API в раннем связывании — это станет понятно после рассмотрения интерфейса прикладного программирования DirectTextToSpeech API.

Из-за включения работы с атрибутами определенным образом увеличится и код инициализации приложения в обработчике `OnCreate` главной формы:

```
procedure TForm1.FormCreate(Sender: TObject);
var
 TTS: TGUID;
 SPD: Cardinal;
begin
 CoCreateInstance(CLSID_VTxt, nil, CLSCTX_LOCAL_SERVER, IID_IVoiceText,
gpIVTxt);
 gpIVTxt.QueryInterface(IID_IVTxtAttributes, gpIVTxtAttr);
```

```
 gpVTxtNotifySink:=TVTxtNotifySink.Create;
 gpIVTxt.Register(nil, 'MyApp', gpVTxtNotifySink, IID_IVTxtNotifySink,
VTXTF_ALLMESSAGES, nil);

 gpIVTxtAttr.TTSModeGet(TTS);
 ComboBox1.Text:=GUIDToString(TTS);
 gpIVTxtAttr.SpeedGet(SPD);
 edSpeed.Text:=IntToStr(SPD);
end;
```

Здесь новая вторая строка реализует получение указателя на интерфейс `IVTxtAttributes`, который мы используем для получения и изменения актуальной информации о речевых атрибутах после регистрации приложения. Конечно же, ко всем переменным в разделе `public` описания формы следует добавить переменную `gpIVTxtAttr`:

```
gpIVTxtAttr: IVTxtAttributes;
```

Так как для изменения речевого движка нам следует знать его ModeID, имеющий тип `TGUID`, то в раскрывающийся список мы введем перечень самых распространенных модулей, для удобства вставляя их названия через разделительную черту "|". В случае выбора модуля из списка нужно всего лишь "вырезать" строку с ModeID и применить его в качестве атрибута.

Аналогично можно поступить с атрибутом скорости, изменение которой поставим в соответствие с событием нажатия клавиши <Enter>.

Замечание

Любопытно, что при использовании объекта `VoiceText` даже в связи с применением атрибутов речи в ходе синтеза происходят некоторые казусы. Например, на моем компьютере программа зависает по команде чтения даже с выбранным родным русским синтезатором (?). Выход состоит в использовании компонента DirectTextToSpeech API, который мне нравится больше, чем Voice Text API.

Демонстрационный проект смотрите в каталоге 03_VTxt_Atr.

Диалоги

В проекте 03_VTxt_Atr реализовано отображение диалоговых окон движка. Для этого сначала нужно объявить переменную интерфейсного типа:

```
gpIVTxtDialogs: IVTxtDialogs;
```

Затем при получении указателей всех интерфейсов аналогичным образом получить интерфейс `IVTxtDialogs` через `QueryInterface`:

```
gpIVTxt.QueryInterface(IID_IVTxtDialogs, gpIVTxtDialogs);
```

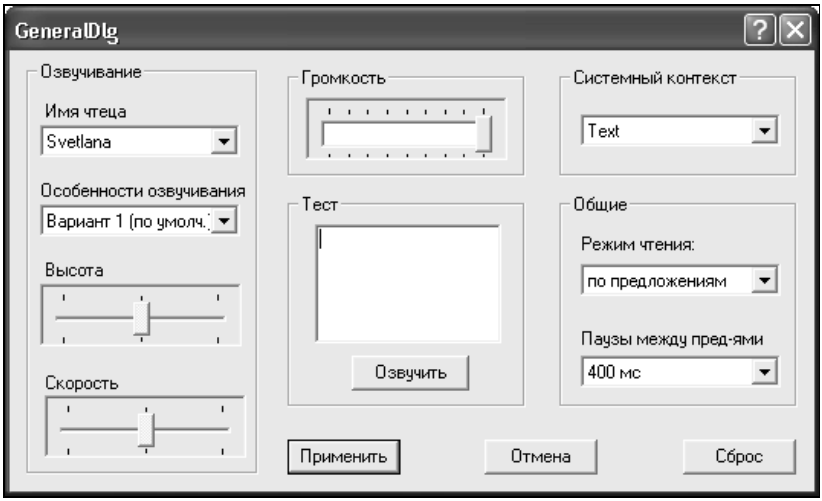


Рис. 3.6. Окно общих параметров (типа General) движков L&H

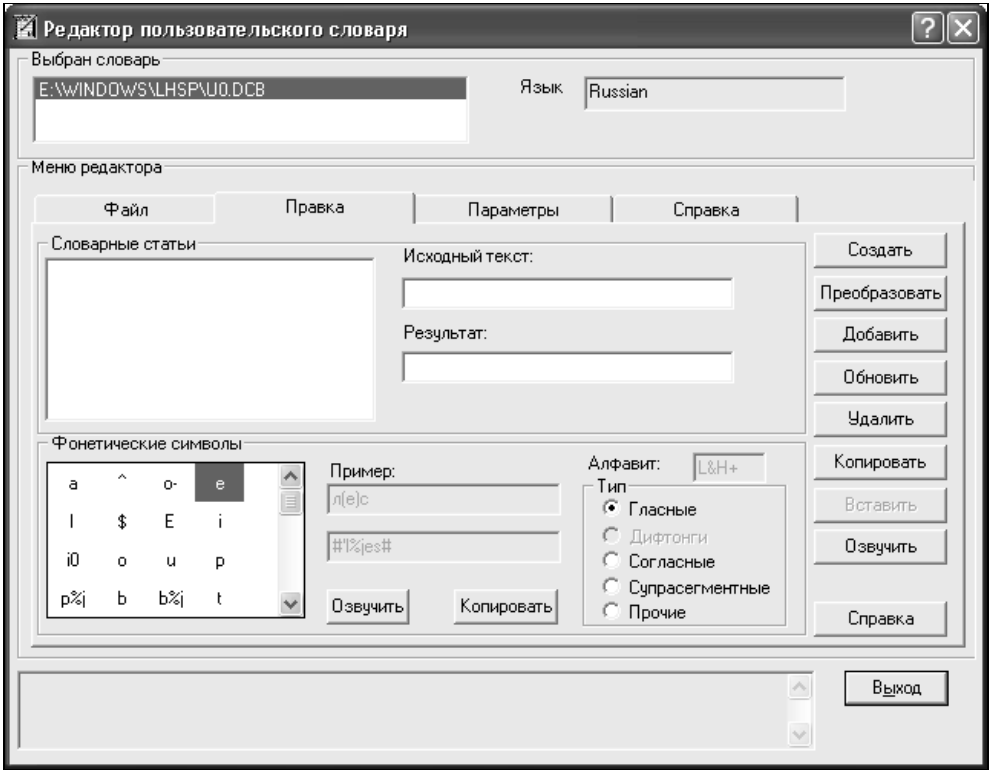


Рис. 3.7. Окно пользовательского словаря (типа Lexicon) движков L&H

После чего запускать требуемые диалоги через методы:

- `AboutDlg`
- `GeneralDlg`
- `LexiconDlg`
- `TranslateDlg`

Тип диалога, запускаемый методом, нетрудно определить из его названия. **About** — отображает информацию о модуле, **General** — окно с основными параметрами модуля (рис. 3.6), **Lexicon** — окно пользовательского словаря данного модуля (рис. 3.7), **Translate** — окно для внесения переопределений (например, таких, чтобы цифры в записи "3.7" читались как "три и семь").

В параметрах методов нужно указывать дескриптор окна, запускающего диалог, и необходимую строку заголовка в диалоге. Однако не все диалоги могут поддерживаться модулем — например, большинство модулей engine не поддерживает диалог **Translate**, а модули Digalo в OEM-версиях движков вообще никаких диалогов, кроме **About**, не поддерживают.

Доступ к объекту *VoiceText* через OLE Automation

Теперь не лишним будет поговорить об использовании объекта `VoiceText` через позднее связывание, т. к. всегда полезно рассматривать различные способы применения одного и того же инструмента.

Как и во всех других приложениях, использующих позднее связывание, мы применим переменную типа `Variant` для хранения указателя на интерфейс `IVoiceText`, а также функцию `CreateOleObject` из модуля `ComObj` для его получения. Сразу приведу небольшой пример:

```
Var
 VTxt: Variant;
...
procedure TfrVTxtAuto.FormCreate(Sender: TObject);
begin
  VTxt := CreateOleObject('Speech.VoiceText');
  VTxt.Register('', Application.ExeName);
end;
```

Самый "сложный" момент здесь — это указание информативного имени класса объекта, которое непросто узнать, но не зная его, нельзя создать объект.

В виде идентификатора приложения при регистрации объекта `VoiceText` в данном случае мы используем имя исполняемого файла, хотя можно было

бы ввести и произвольное значение. Запуск синтеза производится так же, как в случае использования раннего связывания через библиотеку COM:

```
VTxt.Speak(Memo1.Text, 0)
```

Рабочий пример лежит в каталоге 04_VTxt_Auto. Наверное, вас интересует возможность обрабатывать события в позднем связывании. Это также возможно благодаря модулям pas и библиотекам типов tlb от Borland с интерфейсами для реализации некоторых событий.

Для обработки уведомлений нужно подключить специальную библиотеку типов и ее модуль с интерфейсами, включить TLB-файл в исполняемый файл проекта, затем описать класс, реализующий соответствующий Callback-интерфейс.

Итак, в директорию проекта следует приложить файлы VoiceTextAutoVar_TLB.pas и VoiceTextAutoVar.tlb или аналогичные файлы VoiceTextAuto_TLB.pas и VoiceCommandAuto.tlb — они отличаются только названиями. В файле dpr проекта включите директиву присоединения ресурсов в виде TLB-файлов:

```
{$R *.TLB}
```

Затем следует реализовать интерфейс `IVoiceCallback` в каком-либо классе. Так как предком интерфейса является `IDispatch`, то предком класса должен быть `TAutoObject`, а не привычный нам `TInterfacedObject`. Более того, здесь придется вручную обращаться к фабрике классов при инициализации модуля, чтобы инициализировать свойства `DispIntfEntry` и `DispTypeInfo` для нашего объекта, т. к. мы работаем с `IDispatch`. Например, реализация может происходить так, как показано в листинге 3.2 (обратите внимание на то, что она выполнена в виде отдельного модуля, поэтому приходится объявлять модуль главной формы (`Unit1`) для отображения информации о событиях (метод `Log`)).

Листинг 3.2. Модуль, реализующий интерфейс `IVoiceCallback`

```
unit VTxtAutoCallBackU;
{$ifdef CONDITIONALEXPRESSIONS}
{$WARN SYMBOL_PLATFORM OFF}
{$endif}
interface
uses
  ComObj, ActiveX, VoiceTextAuto_TLB, StdVcl;

type
  TVoiceCallback = class(TAutoObject, IVoiceCallback)
```

```
  protected
    procedure SpeakingDone; safecall;
    procedure SpeakingStarted; safecall;
  public
    procedure Initialize; override;
    destructor Destroy; override;
  end;

implementation

uses
 ComServ, Unit1;

procedure TVoiceCallback.Initialize;
begin
  inherited;
  frVTxtAuto.Log('Voice callback создан');
end;

destructor TVoiceCallback.Destroy;
begin
  frVTxtAuto.Log('Voice callback удален');
  inherited;
end;

procedure TVoiceCallback.SpeakingStarted;
begin
  frVTxtAuto.Log('<Старт синтеза речи>');
end;

procedure TVoiceCallback.SpeakingDone;
begin
  frVTxtAuto.Log('<Завершение синтеза речи>');
end;

initialization
  TAutoObjectFactory.Create(ComServer, TVoiceCallback,
Class_VoiceCallback,
```

```
    ciMultiInstance, tmApartment);
  ComServer.UIInteractive := False
end.
```

Все это желательно сделать где-нибудь не в Delphi, а, например, в текстовом редакторе Блокнот (я имею в виду подключение модулей и включение директив, — модуль с реализацией можно, конечно, писать в Delphi). Дело в том, что такие "крупные" подключения Delphi иногда воспринимает неадекватно, тем более если они происходят постепенно, да еще с периодическим сохранением и перекомпиляцией. В итоге проект компилируется нормально, но при запуске сообщает о пропущенной библиотеке типов. Если даже и так собственное приложение, написанное с нуля, у вас не работает, то попробуйте переделать демонстрационный проект.

Теперь при создании объекта `VoiceText` в процедуре `TfrVTxtAuto.FormCreate` формы осталось прописать, кто именно будет отвечать за получение вызовов:

```
procedure TfrVTxtAuto.FormCreate(Sender: TObject);
begin
  VTxt := CreateOleObject('Speech.VoiceText');
  VTxt.Register('', Application.ExeName);

  VTxt.Callback := 'VoiceTextAutoVar.VoiceCallback';
end;
```

Если вы внимательно смотрели описание интерфейса `IVoiceText`, то скорее всего удивитесь, обнаружив свойство `Callback`. В действительности здесь в позднем связывании используется не `IVoiceText`, а интерфейс `IVTxtAuto`, который можно увидеть в модуле VTxtAuto_TLB.pas.

```
IVTxtAuto = interface(IDispatch)
  ['{FF2C7A50-78F9-11CE-B762-00AA004CD65C}']
  procedure Register(const pszSite: WideString; const pszApp:
WideString); safecall;
  procedure Speak(const pszBuffer: WideString; dwFlags: Integer);
safecall;
  procedure StopSpeaking; safecall;
  procedure AudioPause; safecall;
  procedure AudioResume; safecall;
  procedure AudioRewind; safecall;
  procedure AudioFastForward; safecall;
  procedure Set_Callback(const Param1: WideString); safecall;
  procedure Set_Speed(pdwSpeed: Integer); safecall;
  function  Get_Speed: Integer; safecall;
```

```
  procedure Set_Enabled(pdwEnabled: Integer); safecall;
  function  Get_Enabled: Integer; safecall;
  function  Get_IsSpeaking: WordBool; safecall;
  property Callback: WideString write Set_Callback;
  property Speed: Integer write Set_Speed;
  property Enabled: Integer write Set_Enabled;
  property IsSpeaking: WordBool read Get_IsSpeaking;
end;
```

Вы, вероятно, спросите: "Почему "`VoiceTextAutoVar.VoiceCallback`"?" Да потому, что `VoiceTextAutoVar` — это имя класса COM-объекта, предоставляющего интерфейс `IVoiceCallback`, а `VoiceCallback` — это название класса `CoClass`. В данном проекте использовалась библиотека типов VoiceTextAutoVar.tlb, поэтому присутствует имя именно ее объекта. Отличное имя TLB-файла обосновано тем, что при сохранении проекта имя библиотеки типов приобретает название проекта.

Готовый проект находится в каталоге 05_VTxt_AutoNotify.

Работа с объектом *VoiceText* через ActiveX

Использовать объект `VoiceText` можно также и через ActiveX. Рассмотрим это на небольшом примере.

Для использования объекта `VoiceText` через ActiveX первым делом необходимо импортировать соответствующий элемент управления в среду Delphi. В диалоговом окне импорта элемента управления **Import ActiveX** (Импорт ActiveX) (пункт меню **Component | Import ActiveX Control** (Компоненты | Импорт элемента управления ActiveX)) найдите в списке доступных элеметов управления элемент Microsoft Voice Text (если его нет, то нажмите кнопку **Add** (Добавить) и выберите в появившемся окне файл X:\($windows)\speech\VText.dll. Далее установите элемент обычным способом в какой-либо пакет. Рекомендую создать отдельный пакет для Speech-компонентов, т. к. мы еще не раз будем устанавливать различные ActiveX-компоненты для использования объектов SAPI через ActiveX.

На палитре компонентов ActiveX должен появиться новый компонент TextToSpeech с изображением рта. Установив компонент на форму, вы увидите на ней виртуальный рот.

Удобство работы с компонентом по сравнению с прямым обращением к COM-объекту очевидно. Прежде всего, отпадает необходимость получать указатели на интерфейсы. Специальные методы и свойства все делают за вас. Правда, за удобства приходится расплачиваться снижением скорости работы и уменьшением гибкости.

Приведу некоторые основные свойства, методы и события элемента управления TextToSpeech (табл. 3.7). Смысл параметров, не описанных в таблице, станет понятным после ознакомления с *разд. 3.4*.

Таблица 3.7. *Основные свойства и методы элемента управления TextToSpeech*

Объявление	Описание
	Свойства
`property initialized: Integer`	При значении 1 элемент управления инициализирован, при 0 — нет
`property Device: Integer`	Идентификатор устройства вывода
`property Enabled: Integer`	Включает и выключает возможность синтеза речи
`property Speed: Integer`	Скорость речи (количество слов в минуту)
`property TTSMode: WideString`	Идентификатор GUID голоса в строковом формате
`property CountEngines: Integer`	Показывает количество движков в системе
`property IsSpeaking: Integer`	Показывает, производится ли в данный момент чтение или нет
`property LastError: Integer`	Показывает код последней ошибки
	Методы
`procedure Speak(const Text: WideString);`	Запускает чтение текста `Text`, точнее помещает фрагмент текста в очередь чтения
`procedure StopSpeaking;`	Останавливает чтение
`procedure Pause;`	Приостановка синтеза
`procedure Resume;`	Восстановление чтения после паузы
`procedure AboutDlg(hWnd: Integer; const title: WideString);`	Окно **About**
`procedure GeneralDlg(hWnd: Integer; const title: WideString);`	Окно **General**
`procedure LexiconDlg(hWnd: Integer; const title: WideString);`	Окно **Lexicon**
`procedure TranslateDlg(hWnd: Integer; const title: WideString);`	Окно **Tanslate**

Таблица 3.7 *(продолжение)*

Объявление	Описание
	Методы
`function ModeName(index: SYSINT): WideString;`	По порядковому номеру движка возвращает строковое имя голосового режима. Значение включает в себя короткое название фирмы, язык голоса и др. Например: "Adult Male #1 Russian (L&H)". Не путайте с именем диктора (в элементе управления его имя, например, Светлана, Борис, Николай, возвращается через метод `Speaker`). Движки нумеруются с 1, номер последнего определяется по свойству `CountEngines`
`function MfgName(index: SYSINT): WideString;`	Возвращает название фирмы-производителя по номеру движка. Движки нумеруются с 1, номер последнего определяется по свойству `CountEngines`
`function ModeID(index: SYSINT): WideString;`	Возвращает уникальный идентификатор голоса GUID в строковом формате
	События

`Type`
`TTextToSpeechAttribChanged = procedure(Sender: TObject; attrib: Integer) of object;`
`property OnAttribChanged: TTextToSpeechAttribChanged;`
	Происходит при смене атрибутов речи.
	Через параметр `attrib` указывается, какой именно из атрибутов изменился. Возможные значения параметра:
	`IVTNSAC_DEVICE`
	`IVTNSAC_ENABLED`
	`IVTNSAC_SPEED`
	`IVTNSAC_TTSMODE`
	`IVTNSAC_VOLUME`

`Type`
`TTextToSpeechSpeak = procedure(Sender: TObject; const Text: WideString; const App: WideString; thetype: Integer) of object;`
`property OnSpeak: TTextToSpeechSpeak;`
	Возникает, когда в очередь чтения поступает фрагмент текста. Параметр `App` содержит строковый идентификатор приложения, параметр `thetype` показывает тип выражения. Значения параметра `thetype`:
	`VTXTST_COMMAND`
	`VTXTST_NUMBERS`
	`VTXTST_QUESTION`
	`VTXTST_READING`
	`VTXTST_SPREADSHEET`
	`VTXTST_STATEMENT`
	`VTXTST_WARNING`

Таблица 3.7 *(окончание)*

Объявление	Описание
	События
`property OnSpeakingStarted: TNotifyEvent`	Посылается в начале синтеза речи (НЕ фрагмента текста из очереди)
`property OnSpeakingDone: TNotifyEvent`	Посылается по завершению синтеза
`Type` `TTextToSpeechVisual = procedure(Sender: TObject; Phoneme: Smallint; EnginePhoneme: Smallint; hints: Integer; MouthHeight: Smallint; bMouthWidth: Smallint; bMouthUpturn: Smallint; bJawOpen: Smallint; TeethUpperVisible: Smallint; TeethLowerVisible: Smallint; TonguePosn: Smallint; LipTension: Smallint) of object;` `property OnVisual: TTextToSpeechVisual`	Извещает все приложения о текущей читаемой фонеме (первые два параметра), типе выражения (параметр `hints`), о размере и положении губ рта среднестатистического человека, произносящего данную фонему(остальные параметры)

Не будем долго останавливаться на интерфейсе Voice Text API, а перейдем к более гибкому интерфейсу синтеза речи. Полагаю, что после просмотра проекта 06_VTextAX с элементом управления TextToSpeech, позволяющим управлять компонентом Voice Text, все станет понятно.

3.4. Интерфейс DirectTextToSpeech API

Если, вдоволь поиграв с приложениями, построенными на интерфейсе Voice Text API, вы скажете, что вам больше ничего не нужно знать о технологии Speech API, т. к. этого вполне достаточно, чтобы применять ее в своих приложениях, то могу заметить, что самое интересное ждет вас впереди. Подобно первым приложениям с MS Agent, рассмотренным *в гл. 2*, элементарные Speech-приложения впечатляют минимальностью кода и мощностью реализуемых функций. Однако не стоит останавливаться на достигнутом. Ведь если вам действительно нужно будет заняться включением в свою программу функций синтеза речи, которая будет не просто игрушкой, то прикладного интерфейса Voice Text API явно недостаточно.

Несмотря на то, что DirectTextToSpeech API является более низким интерфейсом прикладного программирования, прямой синтез речи (Text-To-

Speech) в нем ничуть не сложнее, чем в Voice Text API. Даже наоборот, именно API обладает большей гибкостью, скоростью и, следовательно, мощными возможностями. Поэтому я предпочитаю использовать этот интерфейс для включения синтеза речи в собственные приложения.

Рассмотрим мы только прямой доступ к интерфейсу в раннем связывании, потому что использование для доступа к DirectTextToSpeech элемента управления DirectSS я считаю проявлением непрофессионализма, т. к. этот способ настолько прост и банален, что и описывать его неинтересно. Тем более что, если вы поймете работу DirectTextToSpeech через COM-объекты, то с ActiveX разберетесь и подавно.

В этом разделе мы начнем знакомиться с объектами Audio Object и Shared Object, а также займемся ручным поиском установленных движков и научимся изменять атрибуты речи на более высоком качественном уровне. Мы поймем, как можно определить речевые возможности синтезатора, такие как диапазоны скорости, тона речи и другие, а также осуществим синтез речи в аудиофайл (!). Надеюсь, я вас уже достаточно заинтриговал, так что приступим.

Основные COM-объекты

Прикладной интерфейс программирования DirectTextToSpeech API иначе называется интерфейсом прямого синтеза речи, т. к. при его использовании приходится все делать руками, в том числе устанавливать соединение с аудиоустройством, производить поиск и выбор необходимого речевого модуля в системе и т. п. Например, в случае с Voice Text API нам не нужно было специально инициализировать звуковое устройство, в которое должны записываться волновые данные. Также можно было пропускать процедуру инициализации речевого модуля — данные по умолчанию загружались из реестра. В DirectTextToSpeech все не так. Тут нам потребуется познакомиться с рядом дочерних объектов из Audio Objects и объектами для работы с движками.

В SAPI имеется ряд базовых аудиообъектов[1] для обращения и отправления звуковых данных на аудиоустройство. Причем под аудиоустройством здесь понимается не только физическое устройство в виде звуковой карты, а еще и волновой файл.

- DirectSound Audio Destination (`AudioDestDirect`)
- Multimedia Audio Destination (`MMAudioDest`)
- File Audio Destination (`AudioDestFile`)

[1] В версии Speech API 4.0 имеется ряд дополнительных объектов, но из-за их более редкого использования они здесь не рассматриваются.

Объект DirectSound Audio Destination самый быстрый, т. к. посылает все аудио "на растерзание" DirectSound API из DirectX. Как следствие, этот объект можно (и нужно, если понадобится синтез речи) использовать только в играх, включающих голосовые технологии. Для обычных приложений по удобству использования и привычности лучше всего подойдет объект Multimedia Audio Destination. Если синтез речи должен осуществляться в звуковой файл, то следует использовать объект File Audio Destination.

Все три названных объекта схожи тем, что поддерживают два основных интерфейса: `IAudio` — для управления внутренним буфером и доступом к атрибутам устройства, и `IAudioDest` — для отправки информации и данных соответствующему объекту Audio Destination. Объект DirectSound Audio Destination, кроме этих интерфейсов, поддерживает еще один интерфейс `IAudioDirect` с единственным методом `Init`. С другими объектами дело обстоит интереснее.

- Объект Multimedia Audio Destination включает еще ряд интерфейсов:
  - `IAudioMultiMediaDevice` — позволяет обращаться к специальным аудиоустройствам;
  - `IAudioDestNotifySink` — через него Multimedia Audio Destination уведомляет TTS-модуль об изменении буфера объекта. Далее обычно модуль уведомляет приложение через свои интерфейсы.
- Объект File Audio Destination, кроме стандартных, включает интерфейсы:
  - `IAudioFile` — для управления файлом;
  - `IAudioFileNotifySink` — для получения уведомлений о состоянии файла;
  - `IAudioDestNotifySink` — для получения движком различного рода уведомлений об изменениях внутреннего буфера File Audio Destination.

Кроме объектов, служащих для доступа к аудиоустройству, необходим COM-объект Engine Object, предоставляющий доступ к речевым модулям. Данный объект не создается вручную через библиотеку COM или другими способами, указатель на его центральный интерфейс `ITTSCentral` возвращается в приложение после выбора синтезатора через объект Text-To-Speech Enumerator — перечислитель так называемых "голосов" (не самих движков, т. к. модуль может иметь несколько разных голосов). Text-To-Speech Enumerator имеет всего два интерфейса: `ITTSEnum` — для перебора всех движков в системе и `ITTSFind` — для поиска движка, исходя из заданных параметров и важности каждого параметра.

Речевой движок (Engine Object), кроме центрального интерфейса должен поддерживать интерфейс для управления атрибутами речи (`ITTSAttributes`), управления диалогами (`ITTSDialogs`), а также обладать парой интерфейсов для послания интерфейсам `ITTSNotifySink` и `ITTSBufNotifySink` уведомлений, информирующих о состоянии чтения и буфера соответственно.

Существует также и перечислитель движков (Engine Enumerator), однако он прямо не используется и создается автоматически — например, тогда, когда происходит поиск голоса через интерфейсы объекта Text-To-Speech Enumerator.

В табл. 3.8 приведены коды ошибок, возвращаемые различными методами COM-объектов прикладного интерфейса DirectTextToSpeech API.

Таблица 3.8. *Коды ошибок методов COM-объектов интерфейса DirectTextToSpeech API*

Константа	Значение	Описание
TTSERR_INVALIDCHAR	$80040208	Текст содержит неверный символ
TTSERR_INVALIDINTERFACE	$80004002	Передан указатель на несуществующий интерфейс
TTSERR_INVALIDKEY	$80040209	Неверный ключ (в методе ITTSCentral.UnRegister)
TTSERR_INVALIDMODE	$80040206	Неверный идентификатор ModelID
TTSERR_INVALIDWINDOW	$8004000F	Неверный дескриптор окна
TTSERR_NOTSUPPORTED	$80004001	Функция не поддерживается
TTSERR_OUTOFDISK	$80040205	Нет свободного места на диске
TTSERR_QUEUEFULL	$8004020A	Очередь текста полна
TTSERR_VALUEOUTOFRANGE	$8000FFFF	Значение выходит за пределы допустимого диапазона
TTSERR_WAVEDEVICEBUSY	$80040203	Звуковое устройство занято
TTSERR_WAVEFORMATNOTSUPPORTED	$80040202	Звуковой формат не поддерживается
TTSERR_NOTPAUSED	$80040501	Синтез не находится на паузе
TTSERR_ALREADYPAUSED	$80040502	Синтез уже приостановлен паузой

Приложение с использованием интерфейса DirectTextToSpeech API

Обобщив материал уже имеющихся сведений, можно обозначить алгоритм работы. Схема работы по реализации синтеза речи через прикладной интерфейс программирования DirectTextToSpeech API такова:

1. Осуществить подключение с аудиоустройству через один из возможных интерфейсов. При необходимости выбрать нужное устройство. Если требуется, то получить указатели на дополнительные интерфейсы.

2. Произвести выбор речевого модуля. Обычно по инициативе нашего Windows-интерфейса на этом этапе происходит поиск всех установленных модулей (синтезаторов), но выбор остается за пользователем, ему все-таки виднее.
3. После того как пользователь выбрал модуль или вы, как разработчик, программно определились с тем, какой из TTS-генераторов будет использоваться для синтеза речи, данный режим нужно применить — при этом возвращается указатель на центральный интерфейс Engine Object.
4. Затем, по усмотрению разработчика, регистрируется объект, реализующий интерфейс `ITTSNotifySink`, который будет получать уведомления о ходе синтеза речи.
5. Наконец, в нужный момент передается буфер текста в очередь на синтез и указатель на объект, реализующий интерфейс уведомлений `ITTSBufNotifySink`.

Давайте разберемся в этой схеме на небольшом примере, однако пока что не будем обрабатывать уведомления и управлять атрибутами. Создадим приложение, в котором можно будет выбирать различные голоса из раскрывающегося меню, а читаемый текст вводить в поле ввода `RichEdit` или `Memo`, и которое будет иметь стандартные кнопки `Speak`, `Pause` и `Stop`.

Первым делом нужно не забыть добавить модули в раздел `uses`. Это самый главный модуль `Speech.pas`, `ActiveX.pas` для обращения к библиотеке COM через процедуру `CoCreateInstance`, модуль `ComObj.pas` для доступа к функции `OleCheck`, а также модуль `mmSystem` для обращения к мультимедийным функциями Windows. Объявим необходимые переменные, например, в разделе `public` главной формы приложения:

```
public
  { Public declarations }
  fITTSCentral: ITTSCentral; {Центральный интерфейс,
    через который производятся все действия с речью}
  fIAMM: IAudioMultimediaDevice;
    {Интерфейс для связи с аудиоустройством}
  aTTSEnum: ITTSEnum;
    {Интерфейс для перебора синтезаторов}
  fpModeInfo: PTTSModeInfo;
    {Указатель на параметры синтезатора, чтобы он всегда был "под рукой"}
  Pause: Boolean;
```

При запуске приложения нужно выполнить подготовительные действия с аудиообъектом, а также перебрать TTS-генераторы и вывести их список.

Запишем это в событие создания формы.

```
procedure TForm1.FormCreate(Sender: TObject);
var
  NumFound : DWord;
  ModeInfo : TTSModeInfo;
begin
  try
    {Инициализация аудиоустройства}
    OleCheck(CoCreateInstance(CLSID_MMAudioDest, Nil, CLSCTX_ALL,
           IID_IAudioMultiMediaDevice, fIAMM));
    OleCheck(fIAMM.DeviceNumSet(WAVE_MAPPER));
  except
  end;
  {Создание перечисляемого объекта для перебора всех движков в системе
   с помощью интерфейса ITTSEnum}
  OleCheck(CoCreateInstance(CLSID_TTSEnumerator, Nil, CLSCTX_ALL,
           IID_ITTSEnum, aTTSEnum));
  OleCheck(aTTSEnum.Reset);//Сбрасываем на первый
  OleCheck(aTTSEnum.Next(1, ModeInfo, @NumFound)); {Получаем первый
движок}
  while NumFound > 0 do
   begin
     ComboBox1.Items.Add(String(ModeInfo.szModeName));
     OleCheck(aTTSEnum.Next(1, ModeInfo, @NumFound)); {Читаем следующий}
   end;
end;
```

Первой строкой кода мы получаем указатель на интерфейс `IAudioMultiMediaDevice`, второй строкой явно указываем устройство, через которое собираемся выводить звук — в данном случае Multimedia Wave-oOut Device. На это указывает константа `WAVE_MAPPER`. Модуль `mmSystem` как раз здесь и пригодился, поскольку константа `WAVE_MAPPER` объявлена именно в нем. Вообще говоря, и в отсутствие второй строчки программа останется работоспособной, однако если в операционной системе по умолчанию установлено другое устройство, то вполне вероятно, что либо синтез речи будет невозможен, либо вы ничего не услышите.

Замечание

Хочу заметить, что в прикладном интерфейсе программирования DirectTextTo-Speech API нам будут часто попадаться структуры и интерфейсы, описанные в

двух синтаксических вариантах. Например, интерфейс `ITTSCentral` может быть описан с разными окончаниями: `ITTSCentralA` для кодировки ANSI, `ITTSCentralW` для кодировки Unicode. Обычно интерфейс или структура в этих вариантах написания без суффикса определена как вариант для ANSI, что наиболее удобно. Поэтому в большинстве случаев этот суффикс не будет указываться — он будет использоваться только тогда, когда написание без суффикса не существует.

Далее создаем экземпляр COM-объекта `TTSEnumerator` (Text-To-Speech Enumerator) и получаем указатель на интерфейс `ITTSEnum` для перебора всех движков в системе. Список методов, принадлежащих интерфейсу `ITTSEnum`, представлен в табл. 3.9.

Таблица 3.9. *Методы интерфейса* `ITTSEnum`

Метод	Описание

`function Reset: HResult; stdcall;`	Сбрасывает перечислитель на первый элемент в списке, т. е. на первый синтезатор.
	Возвращаемые значения:
	`S_FALSE`
	`S_OK`

`function Next(celt: ULONG; var rgelt; pceltFetched: PULONG): HResult; stdcall;`	Отыскивает указанное число синтезаторов (параметр `celt`), помещает их адреса в параметр `rgelt` и в параметре `pceltFetched` указывает число фактически скопированных элементов.
	В коде мы перескакиваем каждый раз на следующий синтезатор, поэтому параметр `celt` равен 1, параметр `rgelt` содержит адрес одного элемента, и при этом параметр `pceltFetched` должен быть больше нуля, если мы еще не добрались до последнего синтезатора в списке
	В случае ошибки возвращает одно из следующих значений:
	`E_INVALIDARG`
	`E_OUTOFMEMORY`
	`E_UNEXPECTED`
	`S_FALSE`

`function Skip(celt: ULONG): HResult; stdcall;`	Пропускает указанное число элементов, при этом виртуальный курсор списка устанавливается перед следующим элементом и для получения его значения нужно пользоваться методом `Next`. Например, если параметр `celt` равен 1 и метод `Skip` применяется после метода `Reset`, то "курсор" окажется перед вторым синтезатором в списке

Таблица 3.9 *(окончание)*

Метод	Описание

`function Clone(out enum: ITTSEnumA): HResult; stdcall;`	Копирует содержимое текущего списка для другого перечислителя, при этом в копии списка "курсор" будет установлен на тот же элемент, что и в исходном списке. Метод удобно применять для сохранения текущей позиции в копии списка, чтобы затем вернуться к ней.
	В случае ошибки возвращает одно из следующих значений:
	`E_INVALIDARG`
	`E_OUTOFMEMORY`
	`E_UNEXPECTED`

`function Select(gModeID: TGUID; out ITTSCentral: ITTSCentralA; IUnknownForAudio: IUnknown): HResult; stdcall;`	Применяет выбранный синтезатор и создает экземпляр объекта Engine Object; указатель на центральный интерфейс помещается в параметр `ITTSCentral`.
	В роли параметра `gModeID` обычно выступает поле `gModeID` структуры `TTSMODEINFO`. В параметре `IUnknownForAudio` указывается адрес выбранного интерфейса для связи с аудиовыводящим устройством.
	В случае ошибки возвращает одно из следующих значений:
	`E_INVALIDARG`
	`E_OUTOFMEMORY`
	`TTSERR_INVALIDMODE`
	`TTSERR_OUTOFDISK`
	`TTSERR_WAVEFORMATNOTSUPPORTED`

Когда список TTS-генераторов выведен, их выбор остается за пользователем. Отслеживать выбор пользователя будем в событии `OnChange` раскрывающегося меню со списком движков. Так как в меню мы выводили только название синтезатора (ModeName), а для выбора нужно использовать уникальный идентификатор ModeID, то придется заново отыскать данный модуль, опираясь на известный порядковый номер движка по значению свойства `ItemIndex` меню `TComboBox`. Меню составлялось на основе интерфейса `ITTSEnum`, поэтому никаких разночтений между списком меню и перечислителем не будет, и следовательно, проще будет воспользоваться методом `Skip` для "перескакивания" на нужный синтезатор в списке, а затем непосредственно через метод `Next` всю информацию о синтезаторе записать в структуру `TTSModeInfo`.

Структура имеет тип `record` и описана следующим образом:

```
TTSMODEINFOA = record
  gEngineID        : TGUID;
  szMfgName        : array [0..TTSI_NAMELEN-1] of AnsiChar;
  szProductName    : array [0..TTSI_NAMELEN-1] of AnsiChar;
  gModeID          : TGUID;
  szModeName       : array [0..TTSI_NAMELEN-1] of AnsiChar;
  Language         : TLanguageA;
  szSpeaker        : array [0..TTSI_NAMELEN-1] of AnsiChar;
  szStyle          : array [0..TTSI_STYLELEN-1] of AnsiChar;
  wGender          : WORD;
  wAge             : WORD;
  dwFeatures       : DWORD;
  dwInterfaces     : DWORD;
  dwEngineFeatures : DWORD;
end;
```

Код события `OnChange` меню:

```
procedure TForm1.ComboBox1Change(Sender: TObject);
var
 NumFound: DWord;
 ModeInfo : TTSModeInfo;{Для хранения информации о текущем движке}
begin
   try
     OleCheck(aTTSEnum.Reset);
     {Перескакиваем на нужный движок}
     OleCheck(aTTSEnum.Skip(ComboBox1.ItemIndex));
     OleCheck(aTTSEnum.Next(1, ModeInfo, @NumFound));
     if assigned(fpModeInfo) then
     {если fpModeInfo не равен nil}
      dispose(fpModeInfo);
     new(fpModeInfo);
     fpModeInfo^:=ModeInfo;
     {загружаем движок:}
     OleCheck(aTTSEnum.Select(fpModeInfo^.gModeID, fITTSCentral,
     IUnknown(fIAMM)));
   except
   end;
end;
```

Напоминаю, что в этом коде используется глобальная переменная для хранения выбранного TTS-режима, являющаяся указателем на структуру `TTSModeInfo`:

```
fpModeInfo: PTTSModeInfo;
```

Поэтому при смене синтезатора переменную приходится очищать и присваивать ей значение заново. Другая глобальная переменная служит для хранения указателя на интерфейс `ITTSCentral`:

```
fITTSCentral: ITTSCentral;
```

Этот указатель инициализируется при выполнении процедуры `Select`.

Вот и все, а теперь осталось запустить чтение буфера текста. Для этого, правда, придется записать текст в специальную структуру `SDATA`, понятную движку. Она содержит указатель на начало данных и размер этих данных.

```
PSData = ^TSData;
{$EXTERNALSYM SDATA}
SDATA = record
  pData: pointer;
  dwSize: DWORD;
end;
TSData = SDATA;
```

Запуск чтения текста, начиная от курсора, из компонента `TRichEdit` может выглядеть так:

```
var
  fSData: TSData;
  BufRich: string;
begin
  if not assigned(fITTSCentral) then
   begin
    ShowMessage('Не выбран движок!');
    exit;
   end;
  BufRich:=copy(RichEdit1.Text, RichEdit1.SelStart,
                  length(RichEdit1.Text)-RichEdit1.SelStart);
  fSData.dwSize := length(BufRich) + 1;
  fSData.pData := pChar(BufRich);
  try
```

```
    OleCheck(fITTSCentral.TextData(CHARSET_TEXT, 0, fSData,
        nil, IID_ITTSBufNotifySink));
  except
  end;
end;
```

Для большей определенности укажу некоторые основные методы интерфейса `ITTSCentral` (табл. 3.10).

Должен обратить ваше внимание на некоторую странность. По идее, мы не рассчитывали на получение уведомлений, поэтому и не создавали никаких объектов, реализующих необходимые интерфейсы. Следовательно, последние два параметра в методе `TextData` должны иметь нулевые значения (указатель на объект `nil` и значение идентификатора интерфейса `GUID_NULL`). Однако со значением `GUID_NULL` метод завершается ошибкой.

Таблица 3.10. *Основные методы интерфейса* `ITTSCentral`

Метод	**Описание**

`function TextData(eCharacterSet: VOICECHARSET; dwFlags: DWORD; dText: SDATA; pNotifyInterface: Pointer; IIDNotifyInterface: TGUID): HResult; stdcall;`	Запускает синтез речи. В виде параметров указываются: текст (`dText`), флаг (`dwFlags`) (определяет обработку речевых тегов — *см. следующий подраздел*) и вид передаваемого текста (`eCharacterSet`). Параметр `eCharacterSet` принимает следующие значения:
	`CHARSET_ENGINEPHONETIC` — набор движковых фонем;
	`CHARSET_IPAPHONETIC` — набор фонем в формате IPA (из международного фонетического алфавита);
	`CHARSET_TEXT` — обычный текст.
	В случае ошибки выполнения метод возвращает одно из следующих значений:
	`E_INVALIDARG`
	`E_OUTOFMEMORY`
	`TTSERR_INVALIDMODE`
	`TTSERR_OUTOFDISK`
	`TTSERR_WAVEFORMATNOTSUPPORTED`

`function AudioPause: HResult; stdcall;`	Делает паузу в синтезе. В случае ошибки возвращает значение `E_OUTOFMEMORY`

Таблица 3.10 *(окончание)*

Метод	Описание

`function AudioResume: HResult; stdcall;`	Восстанавливает синтез речи после паузы. В случае ошибки возвращает одно из следующих значений:
	`E_OUTOFMEMORY`
	`TTSERR_NOTPAUSED`
	`TTSERR_ALREADYPAUSED`

`function AudioReset: HResult; stdcall;`	Прекращает синтез речи. В случае ошибки возвращает значение `E_OUTOFMEMORY`
`function ModeGet(var ttsInfo: TTTSModeInfoA): HResult; stdcall;`	Возвращает информацию о текущем синтезаторе речи в структуру `TTTSModeInfo`

`function Inject(pszTag: PAnsiChar): HResult; stdcall;`	Вставляет управляющий тег в читаемый текст *(см. следующий подраздел)*. В случае ошибки возвращает одно из значений:
	`E_INVALIDARG`
	`E_OUTOFMEMORY`
	`TTSERR_INVALIDCHAR`
	`TTSERR_NOTSUPPORTED`
	`TTSERR_VALUEOUTOFRANGE`

`function Register(pNotifyInterface: Pointer; IIDNotifyInterface: TGUID; var dwKey: DWORD): HResult; stdcall;`	Регистрирует объект, реализующий интерфейс `ITTSNotifySink` для получения уведомлений (для кодировки ANSI — `ITTSNotifySinkA`, для Unicode — `ITTSNotifySinkW`) по указателю на объект и идентификатору интерфейса. Через параметр `dwKey` возвращает ключ, по которому данный объект можно разрегистрировать.
	В случае ошибки возвращает одно из значений:
	`E_INVALIDARG`
	`E_OUTOFMEMORY`
	`TTSERR_INVALIDINTERFACE`

`function UnRegister(dwKey: DWORD): HResult; stdcall;`	Разрегистрирует объект, получающий уведомления интерфейса `ITTSNotifySink` по ключу, полученному в методе `Register`. В случае ошибки возвращает одно из значений:
	`E_INVALIDARG`
	`E_OUTOFMEMORY`
	`TTSERR_INVALIDKEY`

Полный пример проекта находится в каталоге 07_DirectTextToSpeech.

Мы написали приложение, выводящее звуковые данные на устройство, установленное по умолчанию, т. к. не выбирали конкретного устройства через константу `WAVE_MAPPER`. Как правило, профессиональные приложения синтеза речи предоставляют пользовательский интерфейс для выбора устройства, если на компьютере установлено несколько звуковых карт и т. п.

Создать интерфейс для выбора устройств очень просто. Нужно просмотреть весь список доступных устройств, вывести его, а затем при выборе нужного устройства через метод `DeviceNumSet` интерфейса `IAudioMultimediaDevice` сменить устройство по его идентификатору. В этом нам помогут функции Windows из мультимедийного модуля mmSystem.

Суммарное количество аудиовыводящих устройств в системе возвращает функция `waveOutGetNumDevs`. Получить название устройства по его идентификатору `DeviceID` можно функцией `waveOutGetDevCaps`, опять-таки для кодировок ANSI или Unicode нужно подставлять суффикс `A` или `W` соответственно. Все устройства нумеруются с нуля, этот порядковый номер и есть идентификатор устройства `DeviceID`.

```
function waveOutGetDevCaps(uDeviceID: UINT; lpCaps: PWaveOutCaps; uSize:
UINT): MMRESULT; stdcall;
```

Эта функция записывает информацию об устройстве в структуру `PWaveOutCaps`, через которую можно определить название устройства.

```
tagWAVEOUTCAPSA = record
  wMid: Word;                 { manufacturer ID }
  wPid: Word;                 { product ID }
  vDriverVersion: MMVERSION;       { version of the driver }
  szPname: array[0..MAXPNAMELEN-1] of AnsiChar;  { product name(NULL
                                                   terminated string) }
  dwFormats: DWORD;          { formats supported }
  wChannels: Word;            { number of sources supported }
  dwSupport: DWORD;          { functionality supported by driver }
end;
TWaveOutCapsA = tagWAVEOUTCAPSA;
PWaveOutCaps = PWaveOutCapsA;
PWaveOutCapsA = ^TWaveOutCapsA;
```

Вот пример кода, который выводит список в раскрывающееся меню `cbDev`:

```
var
  fDev: PWaveOutCaps;
```

```
  countDev: UINT;
  i: integer;
begin
  countDev:=waveOutGetNumDevs;
  new(fDev);
  cbDev.Items.Add('[Устройство по умолчанию]');
  for i:=0 to countDev-1 do
   begin
    waveOutGetDevCaps(i, fDev, SizeOf(fDev^));
    cbDev.Items.Add(fDev.szPname);
   end;
end;
```

Здесь мы заведомо вводим первой строкой в список запись устройства по умолчанию, чтобы при выборе ее указывать константу `WAVE_MAPPER` вместо конкретного устройства. Теперь идентификатор устройства можно определить по выделенному пункту из списка. Если выделен какой-либо пункт, кроме первого, то идентификатор будет равен `ItemIndex-1`. Если выделен первый пункт, то вместо идентификатора устройства нужно подставлять константу `WAVE_MAPPER`. Например:

```
var
  fIAMM: IAudioMultimediaDevice;
begin
  OleCheck(CoCreateInstance(CLSID_MMAudioDest, Nil, CLSCTX_ALL,
       IID_IAudioMultiMediaDevice, fIAMM));
  if cbDev.ItemIndex=0 then
   OleCheck(fIAMM.DeviceNumSet(WAVE_MAPPER))
  else
   OleCheck(fIAMM.DeviceNumSet(cbDev.ItemIndex-1));
end;
```

Приложение для синтеза речи с использованием выбора устройств реализовано в проекте 08_DTTS_SelectDev (рабочее окно приложения показано на рис. 3.8).

Реализация выбора устройства обязательна для "продвинутых" программ, так что в следующих учебных примерах мы будем опускать эту функцию, чтобы не перегружать код.

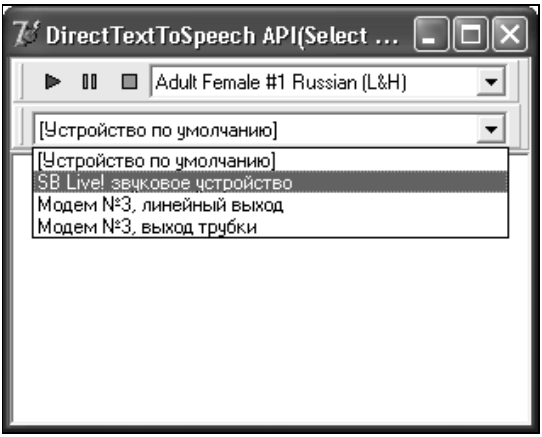


Рис. 3.8. Рабочее окно приложения, реализующего синтез речи с выбором устройств вывода звука

Атрибуты речи

При использовании прямого синтеза речи необходимо управлять различными ее параметрами, такими как скорость речи, тон, громкость. Так же как в прикладном интерфейсе программирования Voice Text API, в DirectTextToSpeech API имеется специальный интерфейс для управления атрибутами синтеза речи, причем он имеет гораздо более широкие возможности. Через него можно изменять не только скорость речи (движок, если помните, меняется через центральный интерфейс), но еще и тембр голоса (Pitch) и громкость (Volume).

Во всей схеме работы с атрибутами речи есть один недостаток, выражающийся лишь в увеличении длины программного кода. Мы уже говорили, что модули синтеза речи разрабатываются различными фирмами, которые ориентируются на спецификацию разработки речевого движка. По существующему стандарту модули синтеза речи должны поддерживать только часть всех возможных интерфейсов и могут не включать некоторые особые свойства (Features) и интерфейсы (Interfaces). Поэтому, получая указатель на структуру `TTSModeInfo`, при поиске движков нелишним будет, во избежание ошибок, проверять поддержку всех необходимых вам функций.

В структуре `TTSModeInfo` есть поле `dwFeatures`, имеющее тип двойного слова. Каждый бит этого параметра отвечает за поддержку той или иной функции движка. Если бит включен, то соответствующая особенность поддерживается. В модуле Speech описаны специальные константы с префиксом `TTSFEATURE_`, каждая из которых обозначает одну из функций (см. табл. 3.11).

Так, логическим сложением поля `dwFeatures` с нужной константой можно определить поддержку требуемой функции. Например:

```
If (fpModeInfo.dwFeatures and TTSFEATURE_PITCH) <> 0 then
  begin
  ...
  end;
```

В табл. 3.11 приведены некоторые константы, представляющие определенные биты свойства `dwFeatures`.

Таблица 3.11. *Описание констант и битов свойства* `dwFeatures`

Константа	Бит	Описание
TTSFEATURE_ANYWORD	0	Синтезатор может читать любое слово
TTSFEATURE_VOLUME	1	Возможность коррекции громкости
TTSFEATURE_SPEED	2	Возможность коррекции скорости
TTSFEATURE_PITCH	3	Возможность коррекции тембра голоса
TTSFEATURE_TAGGED	4	Поддержка речевых тегов *(см. следующий подраздел)*
TTSFEATURE_VISUAL	6	Поддержка информирования о положении рта
TTSFEATURE_WORDPOSITION	7	Поддержка уведомлений о позиции курсора синтеза в читаемом слове
TTSFEATURE_PCOPTIMIZED	8	Синтезатор оптимизирован на синтез через ПК, чтобы давать лучшее качество звука
TTSFEATURE_PHONEOPTIMIZED	9	TTS-модуль оптимизирован на синтез речи через телефон, чтобы использовать частоту дискретизации 8 кГц
TTSFEATURE_SAPI4	16	Поддержка всех возможностей Speech API 4.0, описанных в спецификации

Подобным же образом организуется проверка на поддержку интерфейсов через поле `dwInterfaces`. Соответствующие константы представлены в табл. 3.12.

Таблица 3.12. *Константы и биты свойства* `dwInterfaces`

Константа	Бит	Интерфейс
TTSI_ILEXPRONOUNCE	0	ILexPronounce
TTSI_ITTSATTRIBUTES	1	ITTSAttributes

Таблица 3.12 *(окончание)*

Константа	Бит	Интерфейс
TTSI_ITTSCENTRAL	2	ITTSCentral
TTSI_ITTSDIALOGS	3	ITTSDialogs

В проекте 09_TTSEngineViewer реализован простейший просмотрщик всех синтезаторов речи, установленных в системе. При выборе модуля из списка он показывает исчерпывающую информацию о синтезаторе, голосе, поддерживаемых интерфейсах и функциях (рабочее окно приложения показано на рис. 3.9).

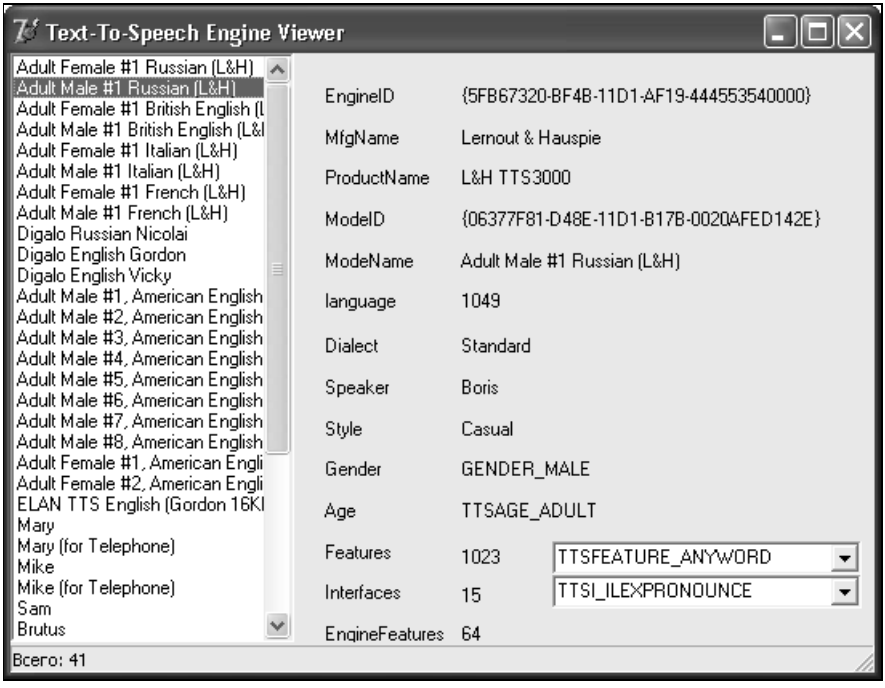


Рис. 3.9. Посмотрите на цифру в строке состояния — это количество движков на компьютере автора

Осуществлять проверку поддержки определенных функций синтезатором можно при его выборе, если право выбирать предоставляется пользователю. Однако можно поступить иначе, а именно, исходя из потребностей вашего приложения можно реализовать программный выбор голоса с учетом требуемых критериев и с учетом важности каждого критерия. Ранее мы попро-

сту перебирали все TTS-модули через интерфейс `ITTSEnum`. Но кроме него объект `TTSEnumerator` имеет еще один интерфейс — `ITTSFind`, обладающий вполне достаточным количеством методов для работы (их два). Один метод предназначен для непосредственного поиска — `Find`, другой для выбора модуля — `Select`. Метод `Select` работает аналогично одноименному методу интерфейса `ITTSEnum`, поэтому поговорим о методе `Find`. Прежде всего рассмотрим его прототип:

```
function Find(var ttsInfo: TTTSModeInfoA; pttsInfoRank: PTTSModeInfoRank;
    var ttsInfoFound: TTTSModeInfoA): HResult; stdcall;
```

Принцип работы метода следующий. В него передается структура `TTTSModeInfoA` с заданными желаемыми характеристиками синтезатора и структура `PTTSModeInfoRank`, содержащая значение уровня важности каждого из параметров. В результате метод выбирает самый подходящий синтезатор и записывает его информацию в параметр `ttsInfoFound` тоже типа `TTTSModeInfoA`. Удобно, не правда ли?

Рабочий пример, реализующий поиск модулей по самым важным критериям, находится в каталоге 10_TTSEngineFinder. В листинге 3.3 приведен код главной формы этого проекта.

Листинг 3.3. Модуль главной формы проекта 10_TTSEngineFinder

```
unit Unit1;


interface


uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, ExtCtrls, StdCtrls, ActiveX, ComObj, Speech, CheckLst, ComCtrls;


const
  InterfacesString : array[0..6] of string = ('TTSI_ILEXPRONOUNCE',
'TTSI_ITTSATTRIBUTES', 'TTSI_ITTSCENTRAL', 'TTSI_ITTSDIALOGS',
'TTSI_ATTRIBUTES', 'TTSI_IATTRIBUTES', 'TTSI_ILEXPRONOUNCE2');
  FeaturesString : array[0..16] of string = ('TTSFEATURE_ANYWORD',
'TTSFEATURE_VOLUME', 'TTSFEATURE_SPEED', 'TTSFEATURE_PITCH',
'TTSFEATURE_TAGGED', 'TTSFEATURE_IPAUNICODE', 'TTSFEATURE_VISUAL',
'TTSFEATURE_WORDPOSITION', 'TTSFEATURE_PCOPTIMIZED',
'TTSFEATURE_PHONEOPTIMIZED', 'TTSFEATURE_FIXEDAUDIO',
'TTSFEATURE_SINGLEINSTANCE', 'TTSFEATURE_THREADSAFE',
'TTSFEATURE_IPATEXTDATA', 'TTSFEATURE_PREFERRED',
'TTSFEATURE_TRANSPLANTED', 'TTSFEATURE_SAPI4');
```

```
type
  TForm1 = class(TForm)
    Label21: TLabel;
    Label7: TLabel;
    Label1: TLabel;
    Label3: TLabel;
    Label15: TLabel;
    Label9: TLabel;
    Label17: TLabel;

    lblanguage: TEdit;{Идентификатор языка}
    lbModeName: TEdit;{Наименование модели движка}
    lbModeID: TEdit;{GUID движка}
    lbMfgName: TEdit;{Фирма-производитель}
    CheckListBox1: TCheckListBox; {Список необходимых интерфейсов}
    CheckListBox2: TCheckListBox; {Список необходимых особенных свойств}
    cbGender: TComboBox; {Пол диктора}

    {С приставкой tr введены элементы TTrackBar, через которые
     устанавливается степень важности соответствующего параметра}
    trMfgName: TTrackBar;
    trModeID: TTrackBar;
    trModeName: TTrackBar;
    trlanguage: TTrackBar;
    trGender: TTrackBar;
    trFeatures: TTrackBar;
    trInterfaces: TTrackBar;

    Button1: TButton; {Кнопка поиска}
    Memo1: TMemo; {Поле для вывода результата поиска}
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    aITTSFind: ITTSFind;
      {Интерфейс для перебора движков}
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  CoCreateInstance(CLSID_TTSEnumerator, Nil, CLSCTX_ALL, IID_ITTSFind,
aITTSFind);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
 TTFindInfo: PTTSModeInfo;
 TTSRank: PTTSModeInfoRank;
 ModeInfo : TTSModeInfo;
 i, j, dwFeatures, dwInterfaces: integer;
 s: string;
begin
 New(TTFindInfo);
 New(TTSRank);

 {******** Заполняем свойства структуры TTFindInfo *********}
 //Переписываем фирму-производитель в свойство szMfgName:
 s:=lbMfgName.Text;
 for i:=0 to TTSI_NAMELEN-1 do
  TTFindInfo.szMfgName[i]:=s[i+1];

 TTFindInfo.gModeID:=StringToGUID(lbModeID.Text);

 //Переписываем наименование модели движка в свойство szModeName:
 s:=lbModeName.Text;
 for i:=0 to length(s) do
  TTFindInfo.szModeName[i]:=s[i+1];

 TTFindInfo.Language.LanguageID:=StrToInt(lblanguage.Text);
 TTFindInfo.Language.szDialect:='Standart';
 TTFindInfo.wGender:=cbGender.ItemIndex;
```

```
//Заполняем свойство dwFeatures на основе установленных флажков
dwFeatures:=0;
for i:=0 to 16 do
 begin
  dwFeatures:=dwFeatures or Integer(CheckListBox2.Checked[i]);
  dwFeatures:=dwFeatures shl 1;
 end;
TTFindInfo.dwFeatures:=dwFeatures;

//Заполняем свойство dwInterfaces на основе установленных флажков
dwInterfaces:=0;
for i:=0 to 6 do
 begin
  dwInterfaces:=dwInterfaces or Integer(CheckListBox1.Checked[i]);
  dwInterfaces:=dwInterfaces shl 1;
 end;
TTFindInfo.dwInterfaces:=dwInterfaces;

{******* Устанавливаем степень важности каждого параметра
             на основе позиции бегунка в TrackBar ******}
TTSRank.dwMfgName:=trMfgName.Position;
TTSRank.dwModeID:=trModeID.Position;
TTSRank.dwModeName:=trModeName.Position;
TTSRank.dwLanguage:=trlanguage.Position;
TTSRank.dwGender:=trGender.Position;
TTSRank.dwFeatures:=trFeatures.Position;
TTSRank.dwInterfaces:=trInterfaces.Position;

{****** Запуск поиска подходящего движка ******}
aITTSFind.Find(TTFindInfo^, TTSRank, ModeInfo);

{****** Вывод результата в Memo1 ******}
Memo1.Lines.Clear;
Memo1.Lines.Add('szModeName: '+StrPas(ModeInfo.szModeName));
Memo1.Lines.Add('LanguageID: '+IntToStr(ModeInfo.Language.LanguageID));
Memo1.Lines.Add('Dialect: '+StrPas(ModeInfo.Language.szDialect));
Memo1.Lines.Add('szMfgName: '+StrPas(ModeInfo.szMfgName));
```

```
Memo1.Lines.Add('szProductName: '+StrPas(ModeInfo.szProductName));
Memo1.Lines.Add('gModeID: '+GUIDToString(ModeInfo.gModeID));
Memo1.Lines.Add('gEngineID: '+GUIDToString(ModeInfo.gEngineID));
Memo1.Lines.Add('szSpeaker: '+StrPas(ModeInfo.szSpeaker));
Memo1.Lines.Add('szStyle: '+StrPas(ModeInfo.szStyle));
 case ModeInfo.wGender of
   0 : Memo1.Lines.Add('wGender: GENDER_NEUTRAL');
   1 : Memo1.Lines.Add('wGender: GENDER_FEMALE');
   2 : Memo1.Lines.Add('wGender: GENDER_MALE');
 end;

 case ModeInfo.wAge of
   1 : Memo1.Lines.Add('wAge: TTSAGE_BABY');
   3 : Memo1.Lines.Add('wAge: TTSAGE_TODDLER');
   6 : Memo1.Lines.Add('wAge: TTSAGE_CHILD');
   14: Memo1.Lines.Add('wAge: TTSAGE_ADOLESCENT');
   30: Memo1.Lines.Add('wAge: TTSAGE_ADULT');
   70: Memo1.Lines.Add('wAge: TTSAGE_ELDERLY');
 end;

 Memo1.Lines.Add('dwFeatures: '+IntToStr(ModeInfo.dwFeatures));
 j := ModeInfo.dwFeatures;
 for i := 0 to 16 do
  begin
   if j and 1 = 1 then
     Memo1.Lines.Add('   '+FeaturesString[i]);
   j := j shr 1;
  end;

 Memo1.Lines.Add('dwInterfaces: '+IntToStr(ModeInfo.dwInterfaces));
 j := ModeInfo.dwInterfaces;
 for i := 0 to 6 do
  begin
   if j and 1 = 1 then
     Memo1.Lines.Add('   '+InterfacesString[i]);
   j := j shr 1;
  end;
```

```
  Memo1.Lines.Add('dwEngineFeatures:
'+IntToStr(ModeInfo.dwEngineFeatures));

  //Устанавливаем фокус на первую строку:
  Memo1.SelStart:=0;
  Memo1.SelLength:=0;
end;

end.
```

Параметр `pttsInfoRank` в методе `Find` интерфейса `ITTSFind` можно не указывать, т. е. оставлять его равным `nil` — в этом случае значения степени важности у каждого параметра структуры `TTTSModeInfoA` будут определяться значениями по умолчанию. Это уменьшит объем кода и обеспечит более удобное применение метода `Find` — без заполнения структуры `PTTSModeInfoRank`, т. к. по умолчанию все параметры имеют собственные приоритеты.

Теперь мы знаем, что перед тем как изменять какие-либо параметры движка, нужно проверить поддержку этих изменений. Далее схема аналогична схеме для интерфейса Voice Text API. Нам нужно получить указатель на интерфейс, отвечающий за атрибуты речи, и изменять в нем необходимые параметры. В прикладном интерфейсе DirectTextToSpeech API эту роль выполняет интерфейс `ITTSAttributes`.

Обычно удобнее предоставлять изменение скорости, тембра и громкости речи через элементы `TrackBar` или `ScrollBar`, так мы поступим в нашем следующем примере. Три компонента `TrackBar` станут управлять этими атрибутами, а над ними в текстовых метках будет указываться максимальное и минимальное значения каждого атрибута. Чтение значений производится тремя методами интерфейса `ITTSAttributes` — `SpeedGet`, `PitchGet` и `VolumeGet`. Изменяются значения атрибутов аналогичными методами с окончанием `Set`. В модуле Speech описаны константы, при передаче которых в виде параметра атрибут изменяется на минимальное или максимальное значение, например, константа `TTSATTR_MINSPEED` изменяет значение скорости речи на минимальное. Во всех остальных константах в названии меняется вторая часть, которая является комбинацией слов `SPEED/PITCH/VOLUME` и `MIN/MAX`.

Хочу обратить ваше внимание на то, что сами константы имеют числовые значения, явно выходящие за пределы допустимого диапазона, именно они и заставляют модуль сменить значение атрибута на верхний или нижний предел. Но надо знать, что другие значения атрибутов, выходящие за область допустимых, вызывают ошибку.

Следует помнить и то, что измененные атрибуты не сохраняются в системном реестре автоматически, как это было в Voice Text API, поэтому, если вам нужно будет восстанавливать настройки в конечном приложении, следует прописывать эту функцию вручную.

В листинге 3.4 приведен пример кода, демонстрирующий получение диапазона возможных значений скорости речи. Код несколько избыточен, однако осуществляет все проверки. Для облегчения понимания даю некоторые комментарии: объект `trSpeed` — класса `TtrackBar` — бегунок для изменения скорости, `lbSpdL` — метка нижнего значения скорости, `lbSpdH` — метка верхнего предела скорости.

Листинг 3.4. Реализация определения минимального и максимального значений скорости речи для выбранного модуля со всеми необходимыми проверками

```
//Проверяем, связана ли переменная fITTSCentral с интерфейсом
if assigned(fITTSCentral) then
 begin
  //Проверяем поддержку атрибутов побитовым умножением
  if(TTSI_ITTSATTRIBUTES and fpModeInfo^.dwInterfaces) <> 0 then
   begin
    try
     //Получаем интерфейс управления атрибутами чтения
     OleCheck(fITTSCentral.QueryInterface(IID_ITTSAttributes,
                  fITTSAttributes));
     //Проверяем поддержку изменения скорости побитовым умножением
     if(TTSFEATURE_SPEED and fpModeInfo.dwFeatures) <> 0 then
      begin
       //Проверяем, поддерживается ли получение значения скорости
       if not(TTSERR_NOTSUPPORTED =
                  fITTSAttributes.SpeedGet(SpeedVoice)) then
        begin
         //Проверяем, поддерживается ли изменение значения скорости
         if not(TTSERR_NOTSUPPORTED = fITTSAttributes.SpeedSet(P)) then
          begin
           trSpeed.Hint:=IntToStr(SpeedVoice);
           {Узнаем минимальную скорость}
           //Заставляем движок изменить значение скорости на
           //минимальное
           fITTSAttributes.SpeedSet(TTSATTR_MINSPEED);
```

```
            //Читаем это значение в переменную MinSpeedVoice
            fITTSAttributes.SpeedGet(MinSpeedVoice);
            lbSpdL.Caption := IntToStr(MinSpeedVoice);
            {Узнаем максимальную скорость чтения (аналогично)}
            fITTSAttributes.SpeedSet(TTSATTR_MAXSPEED);
            fITTSAttributes.SpeedGet(MaxSpeedVoice);
            lbSpdH.Caption := IntToStr(MaxSpeedVoice);
            fITTSAttributes.SpeedSet(SpeedVoice);

            trSpeed.Enabled := True;
            trSpeed.Max := MaxWord;
            trSpeed.Min := 0;
            trSpeed.Max := MaxSpeedVoice;
            trSpeed.Min := MinSpeedVoice;
            trSpeed.Position:= SpeedVoice;
            trSpeed.Frequency := Trunc((MaxSpeedVoice —
                                   MinSpeedVoice) / 10);
          end;
        end;
      end
     else
      begin
       trSpeed.Enabled := False;
      end;
   end;
  except
  end;
 end;
end;
```

Полноценный пример обращения с атрибутами находится в каталоге 11_SpeechAtr, изучите его работу самостоятельно. Должен заметить, что атрибуты речи меняются не мгновенно, а примерно в следующем предложении, если текущее предложение имеет среднюю длину.

Теги

Если вы достаточно хорошо разобрались с первыми программами чтения текстов, то наверняка долго смеялись над произношением — ведь практиче-

ски у всех движков ужасный акцент. Однако, уверяю вас, не нужно разочаровываться: синтез речи весьма продвинутая технология и выдает если еще и не отличные, то уж вполне хорошие результаты. Конечно, заметив, что некоторые слова читаются неправильно, можно кое-что поправить, добавляя нужные слова в пользовательский словарь или — т. к. не все движки поддерживают пользовательские словари — записать эту функцию в вашем приложении вручную. Но что делать, если в некоторых предложениях неправильно ставится логическое ударение, или во время произнесения предложения необходимо динамически менять скорость или тон речи?

Для подобной динамической коррекции созданы так называемые теги (tags). Теги — это текстовые управляющие команды, вставляемые в читаемую текстовую строку, причем они сами не читаются, а подают синтезатору (речевому движку) справочную информацию (команды) по изменению тех или иных параметров чтения текста. Теги обрабатываются речевым синтезатором и, как следствие, не весь спектр тегов Speech API 4.0 наверняка поддерживается всеми движками. Однако наиболее часто употребляемые и распространенные теги, существующие начиная с предыдущих версий SAPI, можно использовать без раздумья. Более того, если введенный тег ошибочен или не поддерживается движком, то модуль прочитает его как набор символов.

Так как теги являются текстовыми командами, которые внедряются в текст, то они должны строиться по определенным правилам, чтобы движок мог отделить команды от основного текста. Приведу правила составления тегов.

- Все теги начинаются и заканчиваются символом наклонной черты ("бэкслэш") влево (\).
- Отдельный бэкслэш не допускается в пределах тега. Чтобы включить бэкслэш в текстовый параметр тега, используйте двойную наклонную черту влево (\\).
- Теги воспринимаются без учета регистра. Например, тег `\Spd=100\` — тот же самый тег, что и `\SPD=100\`.
- Теги пробеловосприимчивы, т. е. не допускают в себе лишних пробелов. Например, теги `\Rst\` и `\ Rst  \` — не одно и то же.

Поддержку тегов синтезатором можно проверить через параметр `dwFeatures` по четвертому биту (установлен в 1) или равенству его значения константе `TTSFEATURE_TAGGED`.

Самые распространенные теги, их синтаксис и описание приведены в табл. 3.13. Примечание к таблице: знак "+" во втором столбце (NS, may not support) означает то, что тег может не поддерживаться синтезатором, а знак "+" в третьем столбце (S4, SAPI 4) — что он новый в SAPI 4.0.

Таблица 3.13. *Основные теги SAPI 4.0*

Тег и синтексис	NS	S4	Описание
\Chr=*string*[[,*string*...]]\			Изменяет выражение голоса. Параметр может состоять из нескольких значений, перечисленных через запятую. Возможные значения: "Angry", "Business", "Calm", "Depressed", "Excited", "Falsetto", "Happy", "Loud", "Monotone", "Perky", "Quiet", "Sarcastic", "Scared", "Shout", "Tense", "Whisper". Тип "Normal" соответствует обычному голосу. Пример: \Chr="*Happy*"\
\Com=*string*\			Комментарий. Комментарии не читаются
\Ctx=*string*\	+		Определяет содержание читаемого текста. Тег может иметь одно из следующих значений: "Address" — адрес и/или телефон; "C" — код на C или C++; "Document" — текст документа; "E-Mail" — E-mail; "Numbers" — число, дата, время и т. д.; "Spreadsheet" — электронная таблица; "Unknown" — неопределенное содержание (значение по умолчанию); "Normal" — нормальный режим чтения
\Emp\	+		Выделяет интонацией слово, стоящее непосредственно за тегом
\Dem\		+	Действие тега обратно предыдущему
\Eng;*GUID*:*command*\			Посылает специальную команду речевому движку с идентификатором GUID. Поддерживаемые команды движка можно узнать в документации к движку. Однако тег все-таки приносит небольшую практическую пользу
\Mrk=*number*\			Ставит в текст закладку с номером number. При синтезе речи в случае обработки уведомлений ITTSBufNotifySink можно будет получить номер закладки в событии BookMark *(см. далее)*. Обычно тег используется для синхронизации каких-либо действий в программе с чтением текста

Таблица 3.13 *(продолжение)*

Тег и синтексис	NS	S4	Описание
\Pau=*number*\			Устанавливает паузу на указанное количество миллисекунд (1/1000 с)
\Pit=*number*\			Указывает тембр голоса в герцах
\Prn=*text=pronunciation*\	+		Включает замену некоторого выражения на необходимое читаемое выражение. Например: \Prn=Юзер=Пользователь\ После этого тега все слова "Юзер" будут читаться как "пользователь". Обратите внимание, что текст выражения пишется без кавычек, т. е. не так, как обычные параметры типа string. Тег \Prn=Юзер\ отменяет сделанную замену. Кроме того, заменяются слова с учетом регистра, т. е. слово "юзер" переопределяться не будет
\Pro=*number*\	+		При значении 0 выключает правила чтения. Текст будет читаться без учета управляющих тегов и правил изменения интонации. Теги будут читаться по символам, а выражения — монотонно. При значение 1 данный режим отключается
\Prt=*string*\	+		Указывает, к какой части речи относится следующее слово. Тег может иметь одно из следующих значений: "Abbr"– сокращение; "Adj" — прилагательное; "Adv" — наречие; "Card" — количественное числительное; "Conj" — объединение слов, например, "M and M"; "Cont" — упрощение, сжатие, например, "can't" вместо "cannot"; "Det" — существительное перед прилагательным; "Interj" — междометие; "N" — существительное; "Ord" — порядковое числительное; "Prep" — предлог; "Pron" — местоимение; "Prop" — существительное в виде имени собственного; "Punct" — пунктуация; "Quant" — существительное, которое выражает количество и предшествует прилагательному, например, "пять зеленых листов"; "V" — глагол
\RmS=*number*\		+	При значении 1 происходит чтение слов по буквам, при значении 0 — по словам

Таблица 3.13 *(продолжение)*

Тег и синтексис	NS	S4	Описание
`\RmW=number\`		+	При значении 1 между словами вставляются небольшие паузы, значение 0 восстанавливает режим без пауз
`\Rst\`		+	Сбрасывает все установленные до него управляющие теги
`\Spd=number\`			Изменяет скорость речи. Параметр имеет значение количества слов в минуту
`\Vce=charact=value[[,charact=value]]\`	+		Заставляет движок изменить синтезатор на другой в соответствии с параметрами. В теге может участвовать несколько параметров с указанными значениями. Например, смена текущего голоса L&H на мужской английский задается так: `\Vce=Gender="Male",language="British English"\` Тег дает команду движку использовать синтезатор английского языка с мужским голосом. Данный тег изменяет значение свойства `ModeID`, т. е. если мы воспользуемся методом `ModeGet` интерфейса `ITTSCentral`, то заметим, что движок будет заменен. В теге возможны следующие параметры: `Language` — язык в специальном формате движка; `Accent` — акцент (например, можно установить язык английским, а акцент французским); `Dialect` — диалект (так же в специальном формате движка); `Gender` — пол (обычно: "Male", "Female" или "Neutral"); `Speaker` — диктор (имя голоса синтезатора); `Age` — возраст (значения: "Baby" (1 год), "Toddler" (3 года), "Child" (6 лет), "Adolescent" (14 лет), "Adult" (20—60 лет), "Elderly" (более 60 лет)); `Style` — стиль голоса (например: "Business", "Casual", "Computer", "Excited" или "Singsong")

Таблица 3.13 *(окончание)*

Тег и синтексис	NS	S4	Описание
\Vol=*number*\			Изменяет громкость речи. Значения от 0 до 65 535 линейно изменяют громкость на обоих каналах. Если нужно изменить громкость на разных каналах, то надо взять число типа двойного слова и в старшую часть (левое слово) записать значение для левого канала, а в младшую (правое слово) — для правого. Например, значение 65 536 устанавливает максимальную громкость на левом канале и минимальную на правом

Теги вставляют непосредственно в читаемую строку, но если вы попробуете прописать теги в коде одного из описанных выше проектов с использованием DirectTextToSpeech API, то у вас ничего не выйдет, потому что в этих приложениях мы не использовали специальный флаг для обработки тегов, который включается в методе `TextData` вторым параметром. Если параметр равен нулю, то теги в строке не обрабатываются и читаются как символы. Если этот параметр равен константе `TTSDATAFLAG_TAGGED` или попросту 1, то синтезатор пытается обрабатывать теги.

Надо заметить, что от внедрения тегов вид текста теряет свою читабельность, да и с точки зрения конечного пользователя, если в программе текст отображается визуально, это не прибавит эстетики вашему приложению. Вставлять теги каждый раз в нужное место довольно неудобно, правда, можно подставлять их перед началом чтения впереди текста автоматически. Вместе с тем теги типа `EMP` нельзя вставить заранее, ведь на то они и теги, чтобы корректировать речь. В конце концов, можно разработать механизм скрытия и отображения тегов подобно тому, как в MS Word скрываются и показываются неотображаемые при печати символы. Но что делать, если нужно внедрять теги динамически в процессе чтения? В этом случае применяется метод `Inject` интерфейса `ITTSCentral`:

```
function Inject(pszTag: PAnsiChar): HResult; stdcall;
```

В виде параметра методу передается обычная строка, включающая только набор речевых тегов. Например:

```
fITTSCentral.Inject(PChar('\Pit='+IntToStr(trPitch.Position)+'\'));
```

С помощью этого метода можно вставлять несколько тегов за один раз.

Замечание

Для вставки тегов через метод `Inject` вовсе не требуется наличие включенного флага в методе `TextData`, эффект наступает и без него.

После вставки тегов через метод `Inject` перемены, как и при внесении изменений атрибутов речи, не сразу применяются в речи, но становятся ощутимы в следующих предложениях.

Нельзя использовать метод `Inject` для вставки тега `Mrk` (закладки) — это приведет к возбуждению исключения. Поэтому закладку нужно вставлять только вручную. Обычно тег закладки используют для синхронизации действий в программе, потому что при достижении синтезатором этого тега приложению посылается уведомление `BookMark` через интерфейс `ITTSBufNotifySink`.

Об уведомлениях в DirectTextToSpeech API читайте в *подразд. "Обработка уведомлений (notification sink)" данного раздела.*

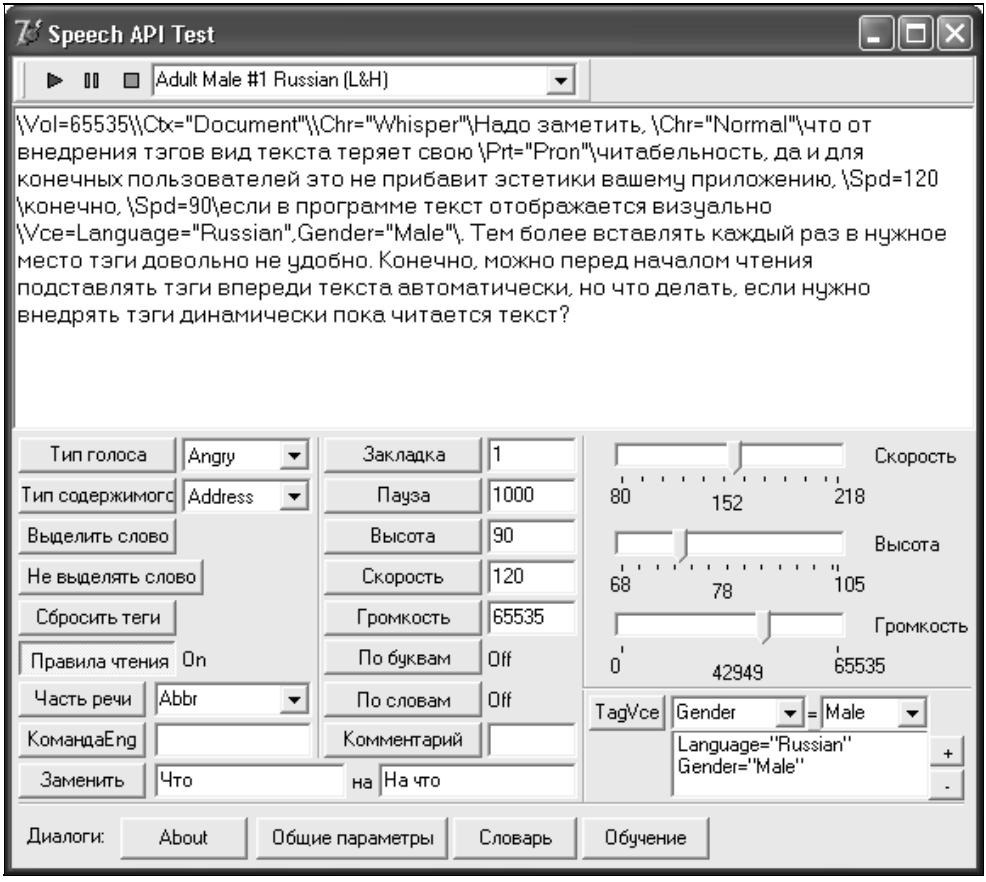


Рис. 3.10. Пример демонстрации тегов

В примере каталога 12_SpeechTegs (рис. 3.10) реализована демонстрация всех самых распространенных тегов SAPI 4.0. С помощью специальных кнопок теги вставляются в поле редактирования, а регуляторами типа `TrackBar`, при изменении их положения, реализуется вставка тегов скорости, тона и громкости через метод `Inject`.

Диалоги

Рассказ был бы неполным без описания интерфейса запуска диалоговых окон. Так как это довольно просто, то скажу лишь, что диалогами в DirectTextToSpeech API занимается интерфейс `ITTSDialogs`. Так же как и интерфейс `IVTxtDialogs` в Voice Text API, интерфейс `ITTSDialogs` имеет четыре метода для отображения соответствующих диалоговых окон: `AboutDlg`, `GeneralDlg`, `LexiconDlg`, `TranslateDlg`. В данные методы также передаются два параметра: дескриптор родительского окна и текст заголовка окна.

Обработка уведомлений (notification sink)

Как уже отмечалось ранее, Engine Object поддерживает два интерфейса уведомлений — `ITTSNotifySink` и `ITTSBufNotifySink`, первый посылает уведомления о состоянии чтения, второй — о состоянии буфера чтения. В табл. 3.14 и 3.15 приведены методы, существующие в этих интерфейсах.

Таблица 3.14. *Методы интерфейса* `ITTSNotifySink`

Метод (Уведомление)	Описание
`function AttribChanged (dwAttribute: DWORD): HResult; stdcall;`	Извещает приложение об изменении атрибутов речи. Параметр `dwAttribute` может иметь одно из следующих значений, символизирующих смену определенного параметра: `TTSNSAC_LANGUAGE` — язык; `TTSNSAC_PITCH` — тембр; `TTSNSAC_REALTIME` — процент процессорного времени, отводящегося на синтез речи (приоритет); `TTSNSAC_SPEED` — скорость речи; `TTSNSAC_VOLUME` — громкость; `TTSNSAC_CUSTOM` — другие параметры речи, а конкретно какие — это можно определить только по изменению полей в структуре `TTSModeInfo` (получить текущие сведения о синтезаторе можно через метод `ModeGet` интерфейса `ITTSCentral`)

Таблица 3.14 *(окончание)*

Метод (Уведомление)	Описание
`function AudioStart(qTimeStamp: QWORD): HResult; stdcall;`	Уведомляет приложение о том, что аудиоданные начинают проигрываться. Вместе с параметром посылается время начала синтеза в байтах, как определено в структуре `FILETIME` в Win32. На практике значение этого параметра в большинстве случаев не используется.
`function AudioStop(qTimeStamp: QWORD): HResult; stdcall;`	Уведомляет приложение о том, что аудиоданные заканчивают проигрываться. В параметре передается время окончания синтеза в байтах
`function Visual(qTimeStamp: QWORD; cIPAPhoneme: AnsiChar; cEnginePhoneme: AnsiChar; dwHints: DWORD; pTTSMouth: PTTSMOUTH): HResult; stdcall;`	Извещает приложение о текущей выводимой фонеме. С параметрами посылается (параметры указываются слева направо): время начала вывода фонемы в байтах, фонема в формате IPA (только Unicode-версии интерфейса `ITTSNotifySinkW`), движковая фонема, тип выражения и указатель на структуру `TTSMouth`, содержащую информацию о положении и позиции рта человека, воспроизводящего фонему. Параметр `dwHints` может содержать одно из следующих значений: `TTSNSHINT_COMMAND` — фонема является частью команды; `TTSNSHINT_EMPHASIS` — фонема — часть, подчеркиваемого выражением слова; `TTSNSHINT_EXCLAMATION` — фонема — часть восклицания; `TTSNSHINT_QUESTION` — фонема — часть вопроса; `TTSNSHINT_STATEMENT` — фонема — часть обычного выражения

Таблица 3.15. *Методы интерфейса* `ITTSBufNotifySink`

Метод (Уведомление)	Описание
`function TextDataStarted (qTimeStamp: QWORD): HResult; stdcall;`	Извещает приложение о том, что синтезатор достиг следующего буфера (фрагмента) текста в очереди. В параметре возвращается время начала следующего фрагмента в байтах

Таблица 3.15 *(окончание)*

Метод (Уведомление)	Описание
`function TextDataDone (qTimeStamp: QWORD; dwFlags: DWORD): HResult; stdcall;`	Извещает приложение о том, что синтезатор достиг конца текущего читаемого буфера. В параметре так же посылается время окончания текущего буфера в байтах. Если параметр `dwFlags` равен 0, то это означает, что весь текст был прочитан, значение `TTSBNS_ABORTED` (или 1) говорит о том, что чтение было прервано методом `AudioReset` центрального интерфейса
`function BookMark(qTimeStamp: QWORD; dwMarkNum: DWORD): HResult; stdcall;`	Уведомляет приложение о том, что синтезатор достиг закладки, вставленной в текст через тег. Возвращает время в байтах, когда закладка была достигнута, и идентификатор закладки
`function WordPosition (qTimeStamp: QWORD; dwByteOffset: DWORD): HResult; stdcall;`	Извещает о номере позиции читаемого слова. Посылает с первым параметром время в байтах и со вторым — смещение в байтах от начала буфера текста. Второй параметр показывает именно смещение в буфере. В принципе, для обычных языков, использующих пробелы между словами, данное уведомление посылается между словами. Причем выражения типа "512" считаются одним словом, а не двумя. Для таких языков, как яитайский или японский, дело обстоит по-другому, т. к. в них нет конкретной семантической границы слова. Пользуясь данным методом (уведомлением), можно легко создавать выделение читаемых слов для визуального контроля

Так же как и для других интерфейсов SAPI, для получения уведомлений от интерфейса DirectTextToSpeech API нужно создать объект, реализующий необходимый интерфейс уведомлений. Объект, реализующий интерфейс `ITTSBufNotifySink`, не нужно регистрировать через специальный метод — интерфейс этих уведомлений не меняется при смене устройства назначения аудиоданных. Поэтому-то все извещения разнесены по нескольким интерфейсам. Так, указатель на `ITTSBufNotifySink` передается с четвертым (предпоследним) параметром в методе `TextData` интерфейса `ITTSCentral` при запуске синтеза речи, а идентификатор интерфейса передается с последним параметром:

```
type
  TTSBufNotifySink = class(TInterfacedObject, ITTSBufNotifySink)
```

```
  ...
  end;

var
  fTTSBufNotifySink: ITTSBufNotifySink;
  fITTSCentral: ITTSCentral;

begin
  try
   fTTSBufNotifySink := TTSBufNotifySink.Create(Self);
   fITTSCentral.TextData(CHARSET_TEXT, 1, SData,
          pointer(fTTSBufNotifySink), IID_ITTSBufNotifySink);
  except
  end;
end;
```

С последними двумя параметрами передается только информация об интерфейсе `ITTSBufNotifySink`. Передавать идентификатор нужно для того, чтобы показать, какой из двух вариантов интерфейса мы собираемся использовать — для кодировки ANSI (`ITTSBufNotifySinkA`, `ITTSBufNotifySink`) или для кодировки Unicode (`ITTSBufNotifySinkW`). Как вы помните, почти для каждого интерфейса или структуры, связанной с передачей символьных данных, существует два варианта их именования с разными суффиксами. Как правило, тип без какого-либо суффикса определен для кодировки ANSI.

Объект, реализующий другую группу уведомлений о состоянии синтеза речи, регистрируется через метод `Register` центрального интерфейса объекта Engine Object:

```
type
  TTSNotifySink = class(TInterfacedObject, ITTSNotifySink)
  ...
  end;
var
  fTTSNotifySink : ITTSNotifySink;
  fdwKey: DWord;
begin
  try
   fTTSNotifySink := TTSNotifySink.Create(Self);
   fITTSCentral.Register(pointer(fTTSNotifySink), IID_ITTSNotifySink,
       fdwKey);
```

```
  except
  end;
end;
```

Через последний параметр метода `Register` возвращается ключ, по которому можно разрегистрировать данный объект через метод `UnRegister`.

Пример приложения, реализующего получение уведомлений, находится в каталоге 13_DTTSNotifySink.

Синтез речи в аудиофайл

Надеюсь, что многих может заинтересовать чтение текстов в аудиофайл, реализованное в некоторых популярных программах чтения голосом. К примеру, очень удобно записать доклад или лекцию в файл, а потом при необходимости прослушать его на ПК или, конвертировав его в формат MP3, воспользоваться для этих целей портативным плеером, незаменимым в дороге. Тем более приятно, что в файл можно записывать речь не в реальном времени, а на порядок увеличив скорость произнесения слов, в результате чего не придется долго смотреть на экран, дожидаясь, пока запишется большой текст.

Между тем, важна не сама эта возможность, т. к. записать звук можно и другими средствами (программами). Однако данная особенность может быть встроена в программу изначально, и тогда пользователь сможет записывать звук средствами самого приложения.

До сих пор мы из всех дочерних объектов Audio Object использовали только Multimedia Audio Destination. Замечу, что для, того чтобы синтез речи производился в файл, нужно заменить интерфейс связи с аудиоустройством на объект File Audio Destination. А вот как именно это сделать и как "выжать" дополнительные опции — мы сейчас и узнаем.

Основной интерфейс управления объекта File Audio Destination (`AudioDestFile`) — `IAudioFile`, через него осуществляются все основные манипуляции с файлом.

```
var
  fIAF: IAudioFile;
```

При инициализации приложения теперь нужно получать указатель не на интерфейс `IAudioMultimediaDevice`, а на интерфейс `IAudioFile`.

```
CoCreateInstance(CLSID_AudioDestFile, Nil, CLSCTX_ALL,
        IID_IAudioFile, fIAF);
```

Далее схема не нарушается. Также осуществляется поиск движка и его выбор, однако при выборе синтезатора методом `Select` мы передаем указатель на интерфейс аудиофайла:

```
aTTSEnum.Select(fpModeInfo^.gModeID, fITTSCentral, IUnknown(fIAF));
```

Затем нужно определить имя конкретного файла, в который будут записываться аудиоданные. Обычно лучше всего делать это перед чтением, а до того предоставить пользователю возможность выбрать имя файла. Реализуется это через интерфейс `IAudioFile` методом `DoSet`. Данный метод выводит из очереди все аудиофайлы и ставит в нее текущий. В параметрах нужно указывать имя файла (в Unicode) и идентификатор файла, который будет посылаться при старте и завершении записи файла в соответствующих уведомлениях. Допустим, в поле редактирования `Edit1` введено имя файла, тогда код, запускающий синтез речи в файл, примет вид:

```
var
  fSData: TSData;
  FlName:  PWideChar;
  fID: Cardinal;
  fIAF: IAudioFile;
begin
  ...
  FlName:=StringToOleStr(Edit1.Text);
  fIAF.DoSet(FlName, hKey);
  fITTSCentral.TextData(CHARSET_TEXT, 0, fSData,
        pointer(fTTSBufNotifySink), IID_ITTSBufNotifySink);
end;
```

После того как будет запущен метод `DoSet`, в файловой системе создастся пустой файл.

> ***Важное замечание***
>
> Интересно заметить, что в исходной англоязычной документации у интерфейса `IAudioFile` нет метода `DoSet`, точнее он называется просто `Set`. В исходном модуле speech.h в интерфейсе объявлен метод с оригинальным называнием. Объясняется это просто — паскалевский речевой модуль содержит ошибки, и это отнюдь не единственная. Поэтому нужно проверять обновления модуля в Интернете.

По завершении чтения вам может показаться странным, что размер файла остался нулевым, однако при закрытии программы он все-таки приобретет свой законный размер. Дело в том, что после завершения чтения нужно удалять файл из очереди, как при работе с обычными файлами, которые

нужно закрывать, чтобы они не испортились; здесь же файл нужно просто освобождать, чтобы можно было к нему обратиться. Делается это методом `Flush`, его следует разместить в процедуре обработки уведомления, например, при получении уведомления `AudioStop` интерфейса `ITTSNotifySink` или `ITTSBufNotifySink.TextDataDone`.

```
function TTSNotifySink.AudioStop(qTimeStamp: QWORD) : HResult;
begin
  result := S_OK;
  fForm.fIAF.Flush;
end;
```

Для того чтобы увеличить скорость записи в файл и тем самым уменьшить время ожидания, нужно применить метод `RealTimeSet` интерфейса `IAudioFile`. В виде параметра здесь передается значение, символизирующее скорость. Так, реальная скорость (однократная) имеет значение 256, или (в шестнадцатеричной форме) $100, двукратная — $200, четырехкратная — $400 и т. д. Странно, что в документации к SAPI — Speech API SDK не указан верхний предел значения скорости, но сразу можно сказать о физическом пределе: это максимальное целое число типа `WORD` — 65 535. Проведенная мною экспериментальная проверка этого параметра закончилась положительно, но скорость была такой же, как шестнадцатикратная, и текст, примерно равный по длине этому абзацу, на моем компьютере записывался чуть более секунды. Тем не менее, реальная скорость записи зависит от скорости процессора, и может получиться так, что на вашем компьютере все скорости, бóльшие восьмикратной или шестнадцатикратной, будут на одно лицо.

При использовании объекта File Audio Destination можно обрабатывать уведомления `IAudioFileNotifySink`. Интерфейс обладает четырьмя методами:

- `FileBegin` — вызывается при старте записи в файл;
- `FileEnd` — вызывается при окончании записи;
- `Posn` — посылает информацию о продвижении процесса записи примерно 16 раз в секунду;
- `QueueEmpty` — вызывается, когда очередь файлов пуста.

Регистрируется объект, реализующий интерфейс через метод `Register` интерфейса `IAudioFile`, в котором передается лишь указатель на интерфейс.

Рабочий проект смотрите в каталоге 14_SpeechToWav.

Полное приложение с интерфейсом DirectTextToSpeech API

Полагаю, читатель смог убедиться в том, что DirectTextToSpeech API — весьма гибкий интерфейс. В следующих разделах мы перейдем уже к распознаванию речи, поэтому напоследок хочется привести в качестве примера наиболее полное приложение, использующее практически все функции, рассмотренные в данном разделе (рис. 3.11).

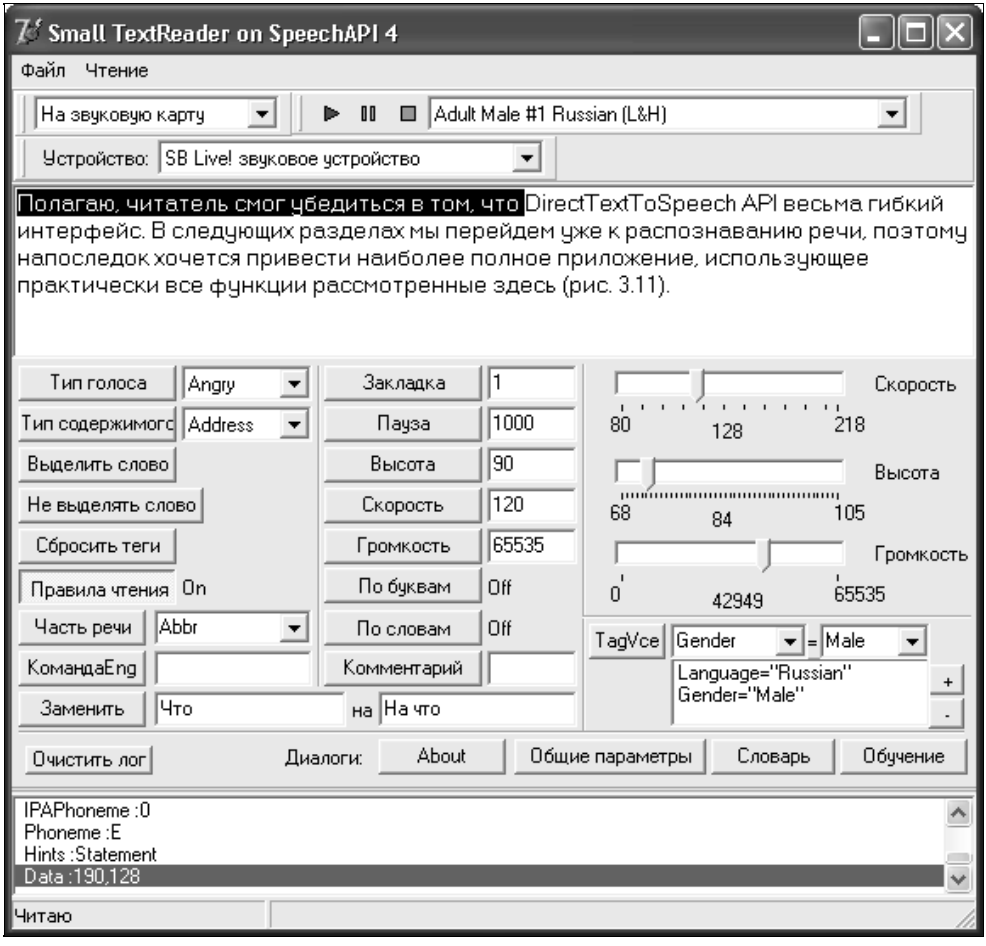


Рис. 3.11. Рабочее окно наиболее полного приложения, реализующего все функции синтеза речи через интерфейс DirectTextToSpeech API

Вывод синтезированной речи в этом примере предусмотрен как на звуковую карту, так и в файл. Также здесь реализован выбор устройства вывода и скорости синтеза речи при записи в файл. При запуске приложения все раскрывающиеся меню инициализируются начальными значениями, чтобы не нарушить потом порядок выбора параметров. Центральная процедура применения всех настроек приложения сосредоточена на событии выбора TTS-модулей — `cbEngine.OnChange`. Она же запускается и при выборе типа вывода (карта-файл), синтезатора и аудиоустройства.

В приложении демонстрируются самые распространенные теги, а также изменение различных атрибутов. Информация обо всех уведомлениях отражается в логе событий.

Проект располагается в каталоге 15_AllSpeechFunc.

3.5. Интерфейс распознавания речи Voice Command API

Сейчас мы переходим к еще более интересному разделу, хотя может быть и менее полезному с практической точки зрения, т. к. технологии распознавания речи находятся на более низком уровне, чем технологии синтеза. Чтение текстов еще можно применить в своей программе (например, для подачи информации в мультимедийных приложениях), а вот использование распознавания речи в повседневной практике — сомнительно. Посудите сами, как будет смотреться со стороны картина — сидит человек и кричит на машину оттого, что она не понимает его слов. Хотя отдельные, заранее предустановленные команды распознаются очень даже неплохо. Разумеется, нужно учитывать и то, что побочные шумы и посторонние звуки типа кашля и "чиха" ухудшают качество распознавания. Но несмотря на эту критику, мне кажется, технологию распознавания речи стоит рассмотреть хотя бы из-за ее непривычности. Ведь не в каждой книге по программированию, а тем более на Delphi, рассказывается, как писать программы, распознающие речь (как, впрочем, и синтезирующие ее).

COM-объекты интерфейса Voice Command API

Интерфейс Voice Command API является интерфейсом высокого уровня. Работа с голосовыми командами напоминает использование обычного меню Windows, только виртуального. Для того чтобы пользоваться этим интерфейсом нужно:

1. Создать экземпляр объекта `VoiceCommand`, чтобы использовать голосовые команды.

2. Выбрать источник звука и идентификатор движка распознавания речи.
3. Обязательно зарегистрировать объект, получающий уведомления, иначе вы не сможете принимать информацию о распознанных словах от модуля распознавания речи.
4. Создать виртуальное меню через специальные методы.
5. Добавить туда необходимые команды.
6. Наконец, активизировать это меню для того, чтобы распознавались его команды.

Объект `VoiceCommand` поддерживает следующие три интерфейса:

- `IVCmdAttributes` — интерфейс управления атрибутами распознавания. Обслуживает выбора типа звукового устройства, микрофона, режима распознавания речи, а также управление включения доступности распознавания;
- `IVCmdDialogs` — интерфейс для отображения диалоговых окон, которые позволяют пользователю настраивать модуль распознавания, а также проводить тренировку голоса;
- `IVoiceCmd` — центральный интерфейс объекта. Через него создаются и удаляются меню; регистрируются объекты, получающие уведомления.

Кроме объекта `VoiceCommand`, в интерфейсе Voice Command API существует еще пара объектов: `VoiceMenu` и `VoiceMenu Enumerator`. Но указатели на их интерфейсы получают не через библиотеку COM. Указатель на интерфейс `IVCmdMenu` управления голосовым меню возвращается при создании меню методом `CreateMenu` интерфейса `IVoiceCmd`. Указатель на интерфейс перечислителя элементов списка команд `IVCmdEnum` объекта `VoiceMenu Enumerator` возвращается при вызове метода `MenuEnum` центрального интерфейса Voice Command Object.

Приложение с объектом *VoiceCommand*

Попробуем создать приложение, распознающее несколько команд. Для полноты его функциональности получим указатели на все интерфейсы, атрибуты, диалоги и т. п. Объявим для этого необходимые переменные:

```
var
  fIVoiceCmd: IVoiceCmd;
  fIVCmdMenu: IVCmdMenu;
  fIVCmdAttrs: IVCmdAttributes;
  fIVCmdDlgs: IVCmdDialogs;
```

В обработчике создания главной формы приложения инициализируем указатели:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  CoCreateInstance(CLSID_VCmd, nil, CLSCTX_ALL,
      IID_IVoiceCmd, fIVoiceCmd);
  fIVoiceCmd.QueryInterface(IID_IVCmdAttributes, fIVCmdAttrs);
  fIVoiceCmd.QueryInterface(IID_IVCmdDialogs, fIVCmdDlgs);
  //Запуск создания меню
end;
```

Теперь нужно создать меню. Лучше всего вынести эти действия в отдельную процедуру. Назовем ее `CreateCommandMenu`. В теле процедуры нужно запустить метод `MenuCreate`, получив при этом указатель на интерфейс `IVCmdMenu`. Затем при помощи этого интерфейса нужно создать все необходимые команды и сделать меню активным. Прототип метода `MenuCreate`:

```
function MenuCreate(pName: PVCmdNameA; pLanguage: PLanguageA;
     dwFlags: DWORD; var IVCmdMenu: IVCmdMenuA): HResult; stdcall;
```

В методе с первым параметром передается указатель `PVCmdNameA`, который ссылается на структуру, содержащую идентификатор приложения и идентификатор меню в приложении. Со вторым параметром передается указатель на структуру `TlanguageA`, содержащую информацию о требуемом языке и диалекте распознавания. При значении `nil` загружается язык модуля распознавания, используемого по умолчанию. Флаг `dwFlags` информирует о типе создания меню (значения для `dwFlags` приведены в табл. 3.16). Последний параметр `IVCmdMenu` — переменная для возвращения указателя на интерфейс `IVCmdMenuA`.

```
var
  VCMDName: TVCMDName;
begin
  StrPCopy(VCMDName.szApplication, ExtractFileName(Application.ExeName));
  VCMDName.szState:='Main';
  OleCheck(fIVoiceCmd.MenuCreate(
    @VCMDName, nil, VCMDMC_CREATE_TEMP, fIVCmdMenu));
  //Добавляем команды
  OleCheck(fIVCmdMenu.Activate(Handle, 0));
end;
```

Таблица 3.16. *Флаги создания меню*

Константа	Значение	Описание
VCMDMC_CREATE_TEMP	$00000001	Создать временное меню с заданным именем. Если меню существует в базе данных VoiceMenu, то метод возвращает ошибку. Временное меню удаляется вместе с объектом VoiceMenu
VCMDMC_CREATE_NEW	$00000002	Создать новое меню с заданным именем. Если меню в базе существует, функция возвращает ошибку. Новое меню сохраняется в базе данных при удалении объекта VoiceMenu
VCMDMC_CREATE_ALWAYS	$00000004	Создает меню с заданным именем. При наличии в базе данных меню с таким же именем оно очищается. При удалении объекта VoiceMenu меню сохраняется в базе
VCMDMC_OPEN_ALWAYS	$00000008	Открывает существующее меню по имени. Если меню не существует, то создается новое пустое меню. Новое меню сохраняется в базе данных при удалении объекта VoiceMenu
VCMDMC_OPEN_EXISTING	$00000010	Открывает существующее меню. Если оно отсутствует в базе, то метод возвращает ошибку

Для добавления команды в меню используется метод Add интерфейса IVCmdMenu:

```
function Add(dwCmdNum: DWORD; dData: SDATA; var dwCmdStart: DWORD):
HResult; stdcall;
```

С первым параметром передается количество добавляемых команд (dwCmdNum). Структура SDATA (второй параметр) должна содержать указатель на список структур VCMDCOMMAND, включающих информацию о голосовых командах. Последний параметр (dwCmdStart) возвращает адрес числа, показывающего номер первой добавленной команды. Все команды нумеруются с 1 и добавляются в конец списка.

Структура VCMDCOMMAND устроена хитро: поле dwSize хранит размер структуры, dwFlags — флаги команды, dwID — ее идентификатор, а вот все остальные поля хранят не сами значения, а смещения, с которых начинается первый символ параметра в поле abData.

```
PVCmdCommand = ^TVCmdCommand;
{$EXTERNALSYM VCMDCOMMAND}
VCMDCOMMAND = record
  dwSize       : DWORD;
```

```
  dwFlags      : DWORD;
  dwID         : DWORD;
  dwCommand    : DWORD;
  dwDescription: DWORD;
  dwCategory   : DWORD;
  dwCommandText: DWORD;
  dwAction     : DWORD;
  dwActionSize : DWORD;
  abData :  record
            end;
end;
TVCmdCommand = VCMDCOMMAND;
```

Нам важно заполнить четыре параметра: идентификатор, название команды (`dwCommand`), ее категорию (`dwCategory`) и описание (`dwDescription`). Название команды — это именно та фраза, которую движок будет пытаться найти в нашей речи. Категория и описание нужны лишь для информативности. Идентификатором также не стоит пренебрегать, он должен содержать уникальное значение команды в меню. С его помощью можно быстро запускать нужные функции приложения, а вот каким образом, об этом мы поговорим чуть позже.

Следующая процедура добавляет команду по ее идентификатору, имени, категории и описанию в заданное меню (глобальная переменная `fIVCmdMenu` типа `IVCmdMenu`). Если реализовать передачу указателя на интерфейс `IVCmdMenu`, то команда будет добавляться в любое меню:

```
procedure AddCommand(ID: Integer; const Command, Category,
    Description: String);
var
  MemReq: Integer;
  CmdCommand: PVCmdCommand;
  SData: TSData;
  Dest: PChar;
  CmdStart: DWord;
begin
  MemReq := SizeOf(TVCmdCommand) + Succ(Length(Command)) +
    Succ(Length(Category)) + Succ(Length(Description));
  CmdCommand := AllocMem(MemReq);
  try
    SData.pData := CmdCommand;
```

```
    SData.dwSize := MemReq;
    CmdCommand.dwSize := MemReq;
    CmdCommand.dwID := ID;

    Dest := PChar(CmdCommand) + SizeOf(TVCmdCommand);
    CmdCommand.dwCommand := SizeOf(TVCmdCommand);
    StrCopy(Dest, PChar(Command));

    Inc(Dest, Succ(Length(Command)));
    CmdCommand.dwCategory := SizeOf(TVCmdCommand) +
         DWord(Succ(Length(Description)));
    StrCopy(Dest, PChar(Category));

    Inc(Dest, Succ(Length(Category)));
    CmdCommand.dwDescription := CmdCommand.dwCommand +
         DWord(Succ(Length(Command)));
    StrCopy(Dest, PChar(Description));

    OleCheck(fIVCmdMenu.Add(1, SData, CmdStart));
  finally
    FreeMem(CmdCommand);
  end;
end;
```

Создадим три команды "Red", "Green" и "Blue", которые будут перекрашивать форму в соответствующие цвета. Для этого сначала нужно вернуться в обработчик создания главной формы приложения, чтобы создать и зарегистрировать объект, получающий уведомления. Это требует создания класса, реализующего интерфейс `IVCmdNotifySink`. В основном нас будут интересовать уведомления `CommandRecognize` и `VUMeter` (см. проект каталога 16_VoiceCommand: в нем продемонстрирована большая часть уведомлений). Первое извещает приложение о распознанной команде, а второе уведомляет об уровне входного звука. Запишем в обработчик события `CommandRecognize` следующий код:

```
function TVCmdNotifySink.CommandRecognize(dwID: DWORD; pvCmdName:
PVCmdNameA; pdwFlags: PDWORD; dwActionSize: DWORD; pAction: pointer;
dwNumLists: DWORD; pszListValues: PAnsiChar; pszCommand: PAnsiChar):
HResult; stdcall;
begin
  Result := S_OK;
```

```
  fForm.AddLog(Format('Команда: App = %s, State = %s, Cmd = %s, Id = %d',
    [pvCmdName.szApplication, pvCmdName.szState, pszCommand, dwId]));
  case dwID of
    1: fForm.Color := clRed;
    2: fForm.Color := clGreen;
    3: fForm.Color := clBlue;
  end
end;
```

Теперь нужно активизировать распознавание речевого движка для объекта `VoiceCommand` методом `EnabledSet` интерфейса атрибутов `IVCmdAttributes` с параметром, равным `True`, и включить распознавание у объекта `VoiceCommand` через метод `AwakeStateSet` того же интерфейса:

```
fIVCmdAttrs.EnabledSet(true);
fIVCmdAttrs.AwakeStateSet(true);
```

Готовое приложение находится в каталоге 16_VoiceCommand (рис. 3.12). Запустите его, установите оба флажка и начинайте произносить команды в микрофон. Полоса индикации `ProgressBar` должна двигаться (как графический эквалайзер на музыкальных центрах), что символизирует повышение уровня громкости звука на входе микрофона. При произнесении команд "Red", "Green" и "Blue" окно должно перекрашиваться в нужный цвет.

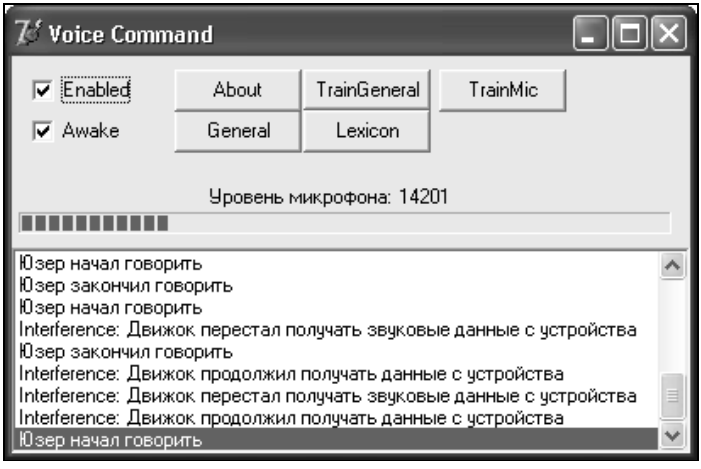


Рис. 3.12. Демонстрационное приложение, распознающее голосовые команды с использованием объекта `VoiceCommand`

Обратите внимание на то, что все манипуляции с атрибутами сохраняются в системном реестре для каждого пользователя.

Списки имен

Иногда могут возникнуть такие ситуации, когда несколько голосовых команд имеют одну одинаковую для всех часть, а другую меняющуюся. Например: "Start Word", "Start Excel" и т. д. Чтобы избежать создания большого числа однотипных команд, применяют списки имен.

На весь набор похожих команд создают одну команду, в которой указывают место в строке команды, где может содержаться выражение из определенного списка. Сам список имеет имя, его также создают и добавляют к этому же меню. Например, для вышеприведенного набора схожих команд формулировка общей команды может быть следующей:

```
'Start <app>'
```

Под именем `app` будет скрываться список из нужных значений. Список имен добавляется в нужное меню посредствам метода `ListSet` под требуемым именем. Вот процедура, которая добавляет в произвольное меню список имен под нужным именем:

```
procedure AddList(Menu: IVCmdMenu; const ListName: String;
  const ListItems: array of String);
var
  List: String;
  I: Integer;
  Data: SData;
begin
  List := '';
  if High(ListItems) < Low(ListItems) then
    Exit;
  for I := Low(ListItems) to High(ListItems) do
    List := List + ListItems[I] + #0;
  Data.pData := @List[1];
  Data.dwSize := Succ(Length(List));
  Menu.ListSet(PChar(ListName), Succ(High(ListItems) - Low(ListItems)),
Data);
end;
```

Из этого кода видно, что значения элементов массива переписываются в одну строку, разделенные символом с нулевым кодом. Затем адрес начала данных массива (т. е. адрес первого символа) заносится в поле указателя структуры `SDATA`, размер строки записывается на единицу больше значения

длины строки. В завершение вызывается метод `LestSet` интерфейса переданного меню.

Пример использования процедуры:

```
AddList(fIVCmdMenu, 'app', ['Word', 'Excel', 'PowerPoint']);
```

Пример готового приложения находится в каталоге 17_VcmdList.

Кроме списков имен, в команды можно вставлять так называемые специальные метки — wild-card, в те места строки команды, где между ключевыми могут находиться любые слова, например:

```
'{wild-card} mail {wild-card} name'
```

Под такую команду подходят фразы типа "Send mail to Fred" или "I want to mail Fred". Однако эту возможность должен поддерживать движок. О том, как проверить, существует ли поддержка, читайте в *разд. 3.7.*

Объект *VoiceCommand* и главное меню

Наверняка, если вы достаточно поупражнялись в создании команд, то задумались над тем, как будет неудобно создавать большое количество команд в конечном приложении, да еще и обновлять их при изменении приложения.

Из этой ситуации существует интересный выход. Если вы работали когда-нибудь с интерфейсом прикладного программирования WinAPI, то помните, как обрабатываются щелчки на кнопках, пунктах меню и т. д. В оконную функцию посылается сообщение `WM_COMMAND` с параметром идентификатора кнопки, по идентификатору определяется, какая кнопка была нажата, и соответственно этому выполняется нужное действие. Так вот, в Delphi пункты меню (класс `TMenuItem`) имеют поле свойства `Command`, и оно как раз содержит такой идентификатор. Если форме, на которой находится окно, послать сообщение `WM_COMMAND` с верхним в списке параметром, равным значению из поля `Command` какого-либо пункта, то выполнится код, записанный в обработчике события `OnClick` этого пункта меню.

Хочу верить, что читатель понял, куда я клоню. Автоматизированное создание команд состоит в упрощении ассоциаций команд главного меню с голосовыми командами. Нужно перебрать все пункты меню и создать команды с идентификаторами, равными свойству `Command`, а затем при распознавании команд нужной форме мы пошлем сообщение, например, так:

```
function TVCmdNotifySink.CommandRecognize(dwID: DWORD; pvCmdName:
PVCmdNameA; pdwFlags: PDWORD; dwActionSize: DWORD; pAction: pointer;
dwNumLists: DWORD; pszListValues: PAnsiChar; pszCommand: PAnsiChar):
HResult; stdcall;
begin
  Result := S_OK;
  Form1.Perform(WM_COMMAND, dwID, 0);
end;
```

В демонстрационном приложении (каталог 18_VCmdMenu) речевые команды строятся по принципу полного пути до пункта меню. Для этого нужно с помощью рекурсивной процедуры перебрать все пункты меню. Вот код этой процедуры:

```
procedure AddMenuCommands(Item: TMenuItem; const ParentPath: String);
var
  I: Integer;
  Path: String;
begin
  Path := ParentPath + StripHotKey(Item.Caption);
  //Рекурсивно просматриваем подменю
  if(Item.Count > 0) then
    for I := 0 to Item.Count - 1 do
      AddMenuCommands(Item.Items[I], Path + #32)
  else
    if(Item.Caption <> '') and(Item.Caption <> '-') then
      AddCommand(Item.Command, Path, ParentPath, Item.Hint);
end;
```

Эта функция использует процедуру `AddCommand` для добавления команд в голосовое меню. Естественно, названия пунктов меню должны быть составлены на языке, который может распознавать движок. Демонстрационное приложение находится в каталоге 18_VcmdMenu. В нем реализовано несложное меню команд, которые вызываются голосом.

Объект *VoiceCommand* в позднем связывании

Интерфейс голосовых команд можно использовать и в позднем связывании — так же, как и в случае с объектом `VoiceText`, здесь нам потребуются дополнительные модули и библиотеки типов (TLB-файлы) для обработки уведомлений. Если при использовании интерфейса синтеза речи Voice Text API можно без них обойтись, то, применяя синтез речи через Voice Command API, нет смысла использовать голосовые команды без поддержки обработки уведомлений. Иначе как мы узнаем о том, что движок распознал голосовую команду? Все модули позаимствованы у фирмы Borland, и они располагаются на прилагаемом компакт-диске. Это файлы VoiceCommandAutoVar_TLB.pas и VoiceCommandAutoVar.tlb. Все действия аналогичны тем, что мы проделывали, работая с Voice Text API. В файле проекта нужно подключить модуль с библиотекой типов и указать директиву для подключения TLB-файлов.

Затем нужно описать класс, который будет реализовывать интерфейс `IListenCallback`, но уже из модуля VoiceCommandAutoVar_TLB.pas.

```
TListenCallback = class(TAutoObject, IListenCallback)
protected
  procedure CommandRecognize(const sCommand: WideString; ID: Integer);
    safecall;
  procedure CommandOther(const sCommand, sApp, sState: WideString);
safecall;
public
  procedure Initialize; override;
  destructor Destroy; override;
end;
```

Инициализация интерфейса голосовых команд может проходить следующим образом:

```
var
  VoiceCmd: Variant;
  CmdMenu: Variant;
  ...
procedure TForm1.FormCreate(Sender: TObject);
begin
  VoiceCmd := CreateOleObject('Speech.VoiceCommand');
  VoiceCmd.Register('');
  VoiceCmd.Callback := 'VoiceCommandAutoVar.ListenCallback';
  CmdMenu := VoiceCmd.MenuCreate(ExtractFileName(Application.ExeName),
    'Main Menu', LANG_ENGLISH, ' ', VCMDMC_CREATE_TEMP);
  CmdMenu.Add(1, 'CmdName', 'CmdCat', 'CmdDesc');
  ...
  CmdMenu.hWndMenu := Handle;
  CmdMenu.Active := True;
  chkAwake.Checked := VoiceCmd.Awake;
end;
```

Полный пример приложения находится в каталоге 19_VCmdAuto.

Объект *VoiceCommand* и ActiveX

Для полноты картины рассмотрим использование объекта `VoiceCommand` через элемент управления ActiveX.

Сначала установим необходимый элемент управления. В окне установки ActiveX он скрывается под именем "Microsoft Voice Command". Если его там

нет, то выберите в папке Speech каталога системы библиотеку Xcommand.dll. После установки элемента управления на палитре ActiveX дополнительно к губам от `VoiceText` появится ухо от `VoiceCommand`. Рассмотрим приложение с голосовыми командами, аналогичное проекту из каталога 19_VCmdAuto. Так как мы работаем с ActiveX, то никаких указателей на интерфейсы получать не надо, достаточно заставить инициализироваться компонент, и он сам соберет нужные данные. Расположим "ухо" `VoiceCommand` на форме (рис. 3.13) и запишем в обработчик создания главной формы приложения следующий код:

```
var
  CmdMenu: Integer;
  ...
procedure TForm1.FormCreate(Sender: TObject);
begin
  Vcommand1.Initialized := Integer(True);
  chkEnabled.Checked := Bool(Vcommand1.Enabled);
  chkAwake.Checked := Bool(Vcommand1.AwakeState);
  CmdMenu:= VCommand1.MenuCreate[ExtractFileName(Application.ExeName),
      'MyMainMenu', VCMDMC_CREATE_TEMP];
  Vcommand1.AddCommand(CmdMenu, cmdID, cmdName, cmdDesc,
     cmdCat, VCMDCMD_CANTRENAME, ' ');
  ...
  Vcommand1.Activate(CmdMenu);
end;
```

Курсивом отмечены параметры идентификатора команды (`cmdID`), названия (`cmdName`), описания (`cmdDesc`) и категории (`cmdCat`).

Для получения голосовых команд надо обработать событие `OnCommandRecognize`:

```
procedure TForm1.Vcommand1CommandRecognize(Sender: TObject; ID: Integer;
  const CmdName: WideString; Flags: Integer; const Action: WideString;
  NumLists: Integer; const ListValues, command: WideString);
begin
  Perform(WM_COMMAND, ID, 0)
end;
```

Для отображения уровня громкости входного звука можно обрабатывать событие `OnVUMeter`. В проекте каталога 20_VCmdAX реализован доступ к объекту `VoiceCommand` через ActiveX (рис. 3.13).

Итак, рассказ об интерфейсе Voice Command API завершен, и мы переходим к еще более заманчивому интерфейсу высокого уровня — Voice Dictation API.

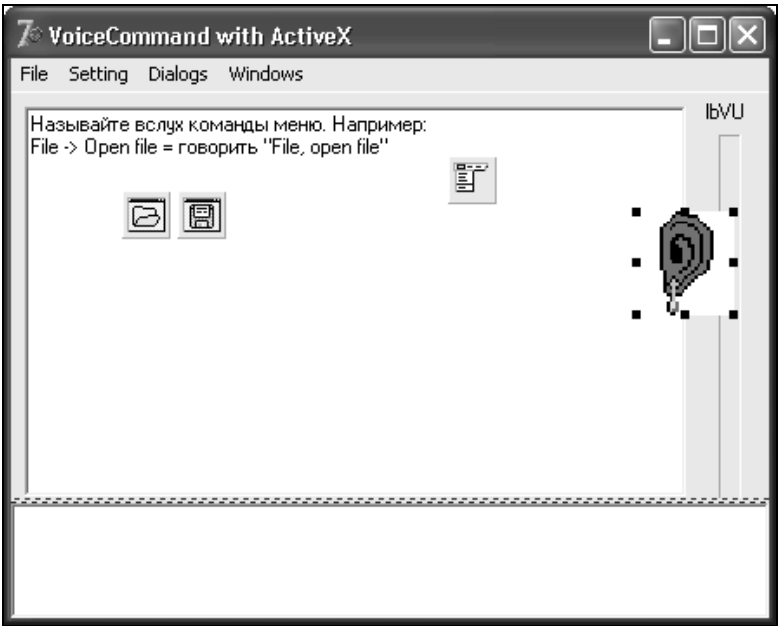


Рис. 3.13. Окно приложения, использующее объект `VoiceCommand` и реализованное через ActiveX (во время разработки)

3.6. Интерфейс распознавания речи Voice Dictation API

Для функционирования примеров этого раздела вам нужно установить движок, поддерживающий диктовку. Обычный шестимегабайтный модуль распознавания речи от Microsoft ее не поддерживает. Поэтому придется устанавливать сорокамегабайтный пакет Speech API SDK Suite 4.0 — самую полную версию SDK, в которой абсолютно точно находится движок, поддерживающий диктовку. Можно "вытащить" его оттуда, он "весит" около 20 Мбайт.

Рассмотрим интерфейс Voice Dictation API лишь для полноты представления о SAPI 4.0, т. к. использование диктовки текста для нас — особенно для программистов — дело, по сравнению с печатаньем на клавиатуре, непосильное. А вот научиться программировать приложения, поддерживающие диктовку, полезно хотя бы для того, чтобы похвастаться перед друзьями ("посмотрите, вот что я могу!").

Основные понятия. Объекты Voice Dictation API

Любую речь можно классифицировать по определенной теме (определяющей и стиль речи). В английской документации для определения подобной классификации предлагается понятие *"топики"* (Topics), именно оно точнее всего характеризует то, что хотели выразить им разработчики SAPI. Например, темами топиков могут быть: "Беседа", "Семья", "Письмо" и т. д. Так вот, каждое приложение, использующее объект `VoiceDictation`, должно классифицировать создаваемый диктовкой документ. Это помогает улучшить качество распознавания речи, сужая множество всех возможных слов и выражений.

Обычно, если приложение создает новый топик, то для этого может потребоваться как время, так и место на жестком диске. Поэтому без острой необходимости не следует создавать новые топики, а в неопределенной ситуации пользоваться топиком General.

Кроме топиков, существуют *сеансы диктовки* — Dictation Session. В каждом приложении необходимо прописать значение этой сессии и связать ее с определенным топиком и дескриптором окна.

Приложение, использующее объект диктовки, может использовать некоторый графический интерфейс для редактирования введенных слов или выражений, своеобразное *окно исправлений* — Correction Window.

Глоссарий (Glossary Entries) может помочь в замене некоторых выражений на другие выражения или на набор символов. Например, когда пользователь говорит "Copyright" система сама заменяет это слово на символ "©", а вместо отображения произнесенного "New-line" осуществляет переход на новую строку.

Команды (Commands) могут использоваться приложением для выполнения не текстовых действий, например, переключения регистра, ввода цифр и т. д.

Кроме того, в диктовке используется *обратная нормализация текста* (Inverse Text Normalization). Модуль диктовки Voice Dictation осуществляет автоматическую расстановку пробелов между словами, включение прописных букв в начале строки и т. д.

У интерфейса высокого уровня Voice Dictation API всего два объекта: `VoiceDictation` и `Results`, через интерфейсы первого объекта осуществляются все действия по распознаванию речи, управлению атрибутами и т. п. Интерфейсы второго объекта используются для обеспечения приложения более детальной информацией, относяйщейся к распознаванию речи.

Объект `VoiceDictation` поддерживает большое количество интерфейсов, и перечислять их все не имеет смысла.

Вот самые основные из них:

- `IVoiceDictation` — основной интерфейс управления объекта `VoiceDictation`, через него выполняется создание сессий диктовки, определяются топики;
- `IVDctNotifySink` — интерфейс используется `VoiceDictation` для уведомления приложения о происходящих событиях;
- `ISRSpeaker` — управляет загрузкой, сохранением и восстановлением информации о дикторе;
- `IVDctAttributes` — управляет атрибутами диктовки, в частности, выбором звукового устройства, микрофона, модуля распознавания речи, активности и доступности распознавания;
- `IVDctCommands` — управляет командами, которые должны быть распознаны в процессе диктовки;
- `IVDctDialogs` — управляет отображением диалоговых окон;
- `IVDctGUI` — этот интерфейс предоставляет доступ к окну коррекции текста (Correction Window);
- `IVDctInvTextNorm` — позволяет приложению устанавливать и управлять обратной текстовой нормализацией, сделанной модулем Voice Dictation;
- `IVDctText` — позволяет приложению получать, устанавливать, перемещать или иначе изменять текст в виртуальном текстовом поле.

Приложение с объектом *VoiceDictation*

Создадим простое приложение, в котором текст при диктовке будет появляться в текстовом поле.

Как обычно, в обработчике события создания формы нужно создать экземпляр необходимого COM-объекта, в данном случае `VoiceDictation`, и получить указатель на интерфейс `IVoiceDictation`.

Затем мы должны зарегистрировать идентификатор приложения для использования, указать идентификатор приложения, выбранный топик и указатель на интерфейс уведомлений (notification sink). Предварительно не забудьте описать класс, реализующий интерфейс уведомлений объекта `VoiceDictation`. В обработке событий нас, в основном, будет интересовать уведомление `PhraseFinish`, посылаемое при уверенном распознавании движком фразы. Уведомление `PhraseStart` посылается, когда движок начинает распознавать аудиоданные, поступающие на вход устройства, после этого может происходить одно или несколько событий `PhraseHypothesis`, посылающих различные варианты распознавания, предполагаемые движком.

Если нет подходящего топика, то его создают перед регистрацией, однако мы будем пользоваться основным топиком — General. После этого получим указатели на различные дополнительные интерфейсы объекта `VoiceDictation`.

Для более качественного распознавания нужно выбрать имя спикера (диктора), т. е. то имя, которое использовалось для тренировки речевого модуля при настройке его на ваш голос. Взгляните на фрагмент кода, приведенный в листинге 3.5, реализующего простейшее приложение с использованием объекта `VoiceDictation`.

Листинг 3.5. Фрагмент кода, реализующий диктовку через `VoiceDictation`

```
unit Unit1;
interface
Uses
 ..., ComObj, ActiveX, Speech;
type
  TVDctNotifySink = class(TInterfacedObject, IVDctNotifySink)
  ...
  End;

var
  fIVDct: IVoiceDictation;
  fIVDctAttrs: IVDctAttributes;
  fIVDctDlgs: IVDctDialogs;
  VDctNotifySink: IVDctNotifySink;

implementation

function TVDctNotifySink.PhraseFinish(dwFlags: DWORD;
      pSRPhrase: PSRPhraseA): HResult; stdcall;
var
  StrPhrase: String;
  dwCnt: Integer;
  PSRW: PSRWord;
Begin
  Result:=S_OK;
  if pSRPhrase = nil then
    Exit;
  StrPhrase:='';
```

```
  dwCnt := pSRPhrase.dwSize - SizeOf(pSRPhrase.dwSize);
  PSRW := @pSRPhrase.abWords;
  while dwCnt > 0 do
   begin
    StrPhrase := StrPhrase + PChar(@PSRW.szWord) + #32;
    Dec(dwCnt, PSRW.dwSize);
    Inc(PChar(PSRW), PSRW.dwSize)
   end;
  fForm.memText.Text := fForm.memText.Text + StrPhrase;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CoCreateInstance(CLSID_VDct, nil, CLSCTX_ALL, IID_IVoiceDictation,
fIVDct);
  fVDctNotifySink:=TVDctNotifySink.Create(Self);
  OleCheck(fIVDct.Register(
       'My Application',
       'General',
       nil,
       nil,
       fVDctNotifySink,
       IID_IVDctNotifySink,
       VCMDRF_ALLMESSAGES));

  fIVDct.QueryInterface(IID_IVDctDialogs, fIVDctDlgs);
  fIVDct.QueryInterface(IID_IVDctAttributes, fIVDctAttrs);

  OleCheck(fIVDctAttrs.SpeakerSet('SpeakerName'));
  OleCheck(fIVDct.Activate(Handle));
  OleCheck(fIVDctAttrs.ModeSet(VSRMODE_CMDANDDCT));
end;
```

Готовое приложение находится в каталоге 21_VoiceDictation (рабочее окно приведено на рис. 3.14).

Если приложение отказывается работать при запуске, то проверьте, все ли компоненты вы установили правильно. В системе обязательно должен быть установлен хотя бы один модуль распознавания речи, поддерживающий диктовку.

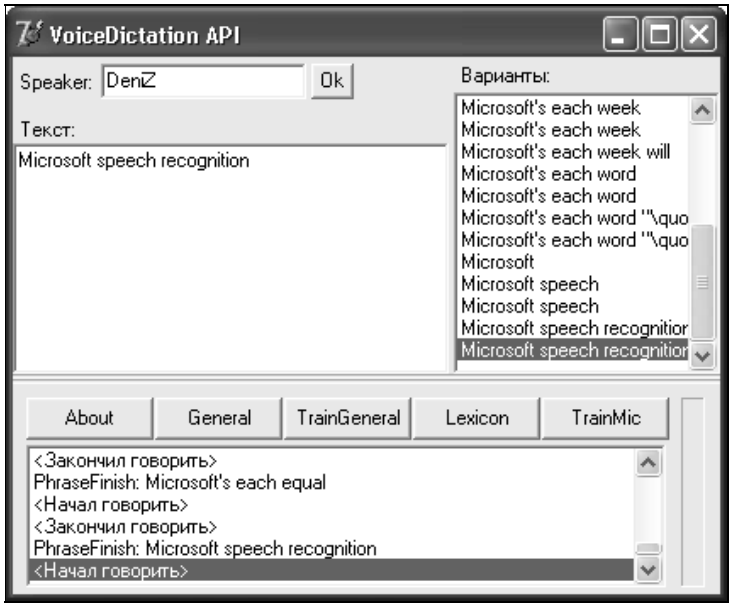


Рис. 3.14. Я сказал "Microsoft Speech Recognition", и посмотрите, что распознал движок (удивительно?)

Учтите, что во время работы между диктовкой и ее завершением может возникать задержка, продолжительность которой зависит от скорости компьютера.

Объект *VoiceDictation* и ActiveX

К интерфейсу Voice Dictation API, как и к другим рассмотренным интерфейсам, существует доступ через элемент управления ActiveX. Для этого в окне установки элементов ActiveX найдите строку "Microsoft Voice Dictation" или выберите в папке Speech файл Vdict.dll. После установки на палитре компонентов ActiveX должна появиться пиктограмма с листочком бумаги и буквой "А" в кружочке.

Разместите компонент на форме приложения, затем проинициализируйте его — либо в Инспекторе объектов (Object Inspector) в среде Delphi, либо программно, присвоив свойству `Initialized` значение 1. Затем следует просто обрабатывать события `OnPhraseFinish` для получения распознанной фразы:

```
procedure TForm1.VdictPhraseFinish(Sender: TObject;
  flags: Integer; const phrase: WideString);
begin
```

```
  if flags and ISRNOTEFIN_RECOGNIZED > 0 then
    mmText.Text:=mmText.Text+phrase;
end;
```

Практически все методы интерфейсов собраны в свойствах и методах элемента ActiveX, поэтому, если вам придется работать с компонентом, то трудностей возникнуть не должно. Полный пример смотрите в каталоге 22_VDctAX.

3.7. Интерфейс синтеза речи DirectSpeechRecognition API

В заключение рассмотрим интерфейс низкого уровня DirectSpeechRecognition API. Как компонент SAPI — DirectTextToSpeech API является более универсальным и гибким инструментом синтеза речи, так и компонент, реализующий прямое распознавание речи (DirectSpeechRecognition API) представляет собой более мощный интерфейс по сравнению с Voice Command API и Voice Dictation API.

Если вы внимательно читали раздел, посвященный компоненту DirectTextToSpeech API *(см. разд. 3.4)*, то для вас не составит особого труда разобраться и с этим низкоуровневым интерфейсом распознавания речи. С помощью прямого распознавания речи можно как распознавать отдельные слова, так и проводить диктовку.

Основные понятия. COM-объекты

Распознавание речи в интерфейсе DirectSpeechRecognition API осуществляется на основе выбранной *грамматики* (Grammar). Грамматика определяет, какие слова, выражения, фразы могут распознаваться движком. Всего существует три вида грамматик:

- Context-free grammar (контекстно свободная грамматика) — определяет набор и порядок слов, которые могут быть распознаны. Распознаются односложные команды и короткие фразы;
- Dictation grammar (грамматика диктовки) — определяет тему диктовки и стиль языка для повышения вероятности распознавания ожидаемых слов;
- Limited-domain grammar (грамматика ограниченного домена) — в данной грамматике слова и выражения жестко не ограничиваются, но в то же время вводятся нестрогие ограничивающие структуры. Этот тип грамматики является чем-то средним между Context-free и Dictation grammar.

Как и в случае с DirectTextToSpeech, при прямом распознавании речи нам нужно обращаться к интерфейсам COM-объектов, обеспечивающим связь с аудиоустройством, — теперь это объекты аудиоисточников звуковых данных.

Из базовых COM-объектов Audio Objects можно выделить для доступа к источникам звука следующие:

- Direct Sound Audio Source (`AudioSourceDirect`) — этот звуковой объект получает аудиоданные от интерфейсов DirectX — DirectSound 5.0 API (или версией выше). Большинство игр должно использовать этот объект, если требуется встроить туда распознавание речи;
- Multimedia Audio Source (`MMAudioSource`) — этот объект получает аудиоданные от интерфейсов MultiMedia API, поэтому его лучше всего использовать в обычных приложениях;
- File Audio Source (`AudioSourceFile`) — этот объект используется для получения приложением аудиоданных из волнового файла.

Далее мы будем рассматривать процесс получения аудиоданных только от Multimedia Audio Source (поскольку он имеет наибольшее практическое применение). Если же все-таки вам потребуется распознать речь, полученную из файла, то, думаю, вы всегда сможете перейти самостоятельно на другой объект источника звука Audio Objects так, как мы это делали в случае с интерфейсом синтеза речи DirectTextToSpeech API.

Итак, Multimedia Audio Source COM Object поддерживает следующие интерфейсы:

- `IAudio` — позволяет объекту `audio-source` управлять внутренним буфером и атрибутами устройства, которое представляет данный объект;
- `IAudioMultiMediaDevice` — предоставляет объекту `audio-source` доступ к специфическим мультимедиа-устройствам;
- `IAudioSource` — получает информацию и данные от объекта `audio-source`;
- `IAudioSourceNotifySink` — уведомляет даижок распознавания речи об изменениях на внутреннем буфере источиника аудиоданных.

Связавшись с аудиоустройством, нужно воспользоваться интерфейсами объекта `SREnumerator` (Speech Recognition Enumerator), чтобы найти необходимый модуль распознавания речи. Так же как и `TTSEnumerator` (Text-To-Speech Enumerator), перечислитель движков распознавания речи имеет два аналогичных интерфейса: `ISREnum` и `ISRFind` — для просмотра всех движков и поиска модуля по заданным критериям соответственно.

Управлением атрибутами речевого модуля, диалогами, загрузкой грамматики, регистрацией уведомлений занимается объект `Engine`, указатель на который также передается в приложение при выборе необходимого модуля мето-

дами интерфейсов объекта `SREnumerator`. Этот объект поддерживает интерфейсы: `ISRCentral`, `ISRAttributes`, `ISRDialogs`, `ISRNotifySink`, `ISRSpeaker`.

Через интерфейс специального объекта `Grammar` происходит управление различными типами грамматик, а также получением уведомлений о распознанных словах и фразах. Объект грамматики поддерживает следующие интерфейсы:

- `ISRGramCFG` — для контекстно-свободной грамматики;
- `ISRGramDictation` — для грамматики диктовки;
- `ISRGramCommon` — для управления различными типами грамматик, активизированием, деактивизированием и т. д.;
- `ISRGramInsertionGUI` — для отображения маленького окна состояния рядом с курсором в поле ввода;
- `ISRGramNotifySink` — для уведомления приложения о различных событиях, связанных с распознаванием фраз речи из грамматики.

Как правило, грамматика представляет собой обычный текст определенной структуры, наподобии ini-файла. Загружать ее можно непосредственно из текстовой строки или из файла.

Кроме рассмотренных основных объектов Speech API в этом разделе мы будем использовать объект `Grammar Compiler` из Speech Tools API, который появился только в SAPI 4.0. Применяется он для преобразования исходного удобочитаемого файла грамматики в вид, понятный движку. Поддерживает этот объект всего один интерфейс — `ISTGramComp`.

Создаем приложение с интерфейсом DirectSpeechRecognition API

Приступаем к программированию. Алгоритм состоит в следующем.

1. Обращаемся к устройству, являющемуся источником аудиоданных, через выбранный интерфейс. В нашем случае это Multimedia Audio Source.
2. Осуществляем поиск модулей распознавания речи по определенным критериям. Если нужна поддержка контекстно-свободной грамматики, то выбираем именно такие движки. При случае с диктовкой нужно выбирать только те движки, которые поддерживают грамматику диктовки. Выдаем список на выбор пользователю или выбираем сами программным путем.
3. Применяем выбор и получаем указатель на центральный интерфейс движка `ISRCentral`.
4. Загружаем необходимую грамматику в объект `Grammar Compiler` и компилируем ее.

5. Создаем объект, реализующий интерфейс уведомлений `ISRNotifySink` для получения информации о состоянии распознавания речи и регистрируем его через метод `Register` центрального интерфейса объекта `Engine`.
6. Создаем объект, реализующий интерфейс `ISRGramNotifySink` для получения уведомлений о распознанных фразах, и загружаем скомпилированную грамматику в движок через объект `Grammar Compiler`, при этом передавая указатель на интерфейс уведомлений. Взамен получаем указатель на интерфейс `IUnknown` объекта `Grammar`.
7. Через возвращенный указатель получаем указатель на интерфейс управления грамматикой `ISRGramCommon`.
8. Через `ISRGramCommon` активизируем движок на распознавание речи по загруженной грамматике.

Схема сложная, но разобраться с ней будет проще, если рассмотреть пример. Данная схема работает, когда нужно использовать грамматику из обычного текстового файла. Для того чтобы преобразовать ее в форамат, понятный движку, мы ее компилируем. Если грамматика задана в нормальной форме, то данная схема может отличаться немного в последних пунктах (п. 4 не будет, а п. 6 упростится), т. к. отпадает необходимость в преобразовании грамматики. В этом случае грамматика передается через метод `GrammarLoad` центрального интерфейса, через него также возвращается указатель на интерфейс `IUnknown` объекта `Grammar`, далее следуют п. 7 и 8.

Как всегда, сначала объявляем необходимые модули: это `ComObj`, `ActiveX`, `mmSystem` и `Speech`. Затем вводим все необходимые переменные — их будет очень много, поэтому привожу все с описанием:

```
var
  fISRCentral: ISRCentral; {Центральный интерфейс,
    через который производятся все действия с речью}
  fIAMMS: IAudioMultiMediaDevice;
    {Интерфейс для связи с аудиоустройством}
  aSREnum: ISREnum;
    {Интерфейс для перебора движков}
  sModeInfo: PSRModeInfo;
    {Указатель на параметры движка}
  SRGramUnk: IUnknown;
    {Указатель IUnknown объекта Grammar после загрузки грамматики}
  fISRGramCommon: ISRGramCommon;
    {Интерфейс управления грамматикой}
  fGramNotifySink: ISRGramNotifySink;
    {Интерфейс получения уведомлений о распознанных фразах}
```

```
  fISRNotifySink: ISRNotifySink;
   {Для получения информации о состоянии распознавания}
  p_dwKey: THandle; {Ключ, возвращаемый после регистрации ISRNotifySink}
  fISTGramComp: ISTGramComp;
   {Интерфейс объекта Grammat Compiler для преобразования грамматики}
  GramErr: PWideChar; {Для получения сообщений после компиляции грамматики}
  Size: DWord; //Размер этого сообщения
```

Для поиска всех устройств источников звука используем мультимедийные функции системы, объявленные в модуле `mmSystem`. Количество таких устройств возвращает функция `waveInGetNumDevs`. Устройства нумеруются с нуля, поэтому цикл вывода названий устройств будет работать, пока значение счетчика меньше или равно $n - 1$. Выводить список будем в раскрывающееся меню `cbDevIn` в блоке обработки события создания формы приложения:

```
var
  fDevIn: PWaveInCaps;
  countDevIn: UINT;
  i: integer;
  ...
begin
  countDevIn:=waveInGetNumDevs;
  new(fDevIn);
  cbDevIn.Items.Add('[Устройство по умолчанию]');
  for i:=0 to countDevIn-1 do
   begin
    waveInGetDevCaps(i, fDevIn, SizeOf(fDevIn^));
    cbDevIn.Items.Add(fDevIn.szPname);
   end;
  cbDevIn.ItemIndex:=0;
  ...
end;
```

Затем создаем объект для связи с аудиоисточником:

```
try
  CoCreateInstance(CLSID_MMAudioSource, Nil, CLSCTX_ALL,
         IID_IAudioMultiMediaDevice, fIAMMS);
  OleCheck(fIAMMS.DeviceNumSet(WAVE_MAPPER));
except
end;
```

Теперь создаем объект-перечислитель движков распознавания речи и выводим в список раскрывающегося меню `cbSREngine` только те движки, которые поддерживают context-free-грамматику:

```
CoCreateInstance(CLSID_SREnumerator, Nil, CLSCTX_ALL,
       IID_ISREnum, aSREnum);
aSREnum.Reset;//Сбрасываем на первый
aSREnum.Next(1, fsrModeInfo, @NumFound); {Получаем первый движок}
while NumFound > 0 do
 begin
   if fsrModeInfo.dwGrammars and SRGRAM_CFG > 0 then
     cbSREngine.Items.Add(String(fsrModeInfo.szModeName));
   aSREnum.Next(1, fsrModeInfo, @NumFound); {Получаем остальные}
 end;
```

Обратите внимание, что функционирование интерфейса `ISREnum` происходит аналогично интерфейсу `ITTSEnum` в DirectTextToSpeech API.

Осталось предоставить выбор пользователю, после чего производим оставшиеся действия. Надо только не забыть реализовать интерфейсы `ISRNotifySink` и `ISRGramNotifySink`. Методы первого приведены в табл. 3.17, второго — в табл. 3.18.

Таблица 3.17. *Методы интерфейса* `ISRNotifySink`

Метод (Уведомление)	Описание
`function AttribChanged (dwAttribute: DWORD): HResult; stdcall;`	Уведомляет об изменениях атрибутов распознавания. Параметр может иметь одно значение, обозначающее смену определенного атрибута. Вот некоторые его значения: `ISRNSAC_AUTOGAINENABLE` — включена автокорректировка мощности входного звука; `ISRNSAC_ECHO` — уровень эха; `ISRNSAC_ENERGYFLOOR` — порог шума; `ISRNSAC_MICROPHONE` — текущий микрофон; `ISRNSAC_REALTIME` — процент процессорного времени, которое движок может использовать в продолжение постоянной речи; `ISRNSAC_SINKFLAGS` — флаги уведомлений; `ISRNSAC_SPEAKER` — имя диктора; `ISRNSAC_THRESHOLD` — порог достоверности распознанной фразы; `ISRNSAC_TIMEOUT` — продолжительность таймаута для завершения фраз

Таблица 3.17 *(окончание)*

Метод (Уведомление)	Описание
`function Interference (qTimeStampBegin: QWORD; qTimeStampEnd: QWORD; dwType: DWORD): HResult; stdcall;`	Уведомляет приложение о том, что движок не смог распознать фразу по одной из следующих причин (флаг может иметь одно из следующих значений): `SRMSGINT_NOISE` — большой фоновый шум; `SRMSGINT_NOSIGNAL` — движок не может фиксировать сигнал; `SRMSGINT_TOOLOUD` — слишком высокий уровень громкости; `SRMSGINT_TOOQUIET` — слишком низкий уровень громкости; `SRMSGINT_AUDIODATA_STOPPED` — движок перестал получать аудиоданные; `SRMSGINT_IAUDIO_STOPPED` — закрыто устройство получения аудиоданных; `SRMSGINT_AUDIODATA_STARTED` — движок начал получать аудиоданные; `SRMSGINT_IAUDIO_STARTED` — открылось устройство получения аудиоданных
`function Sound (qTimeStampBegin: QWORD`	Движок обнаружил звук
`function UtteranceBegin (qTimeStampBegin: QWORD): HResult`	Посылается, когда движок определил начало подачи фразы
`function UtteranceEnd (qTimeStampBegin: QWORD`	Посылается, когда движок определил конец подачи фразы
`function VUMeter (qTimeStampBegin: QWORD`	Уведомляет об уровне громкости звука в цифровом потоке

Таблица 3.18. *Основные методы интерфейса* `ISRGramNotifySink`

Метод (Уведомление)	Описание
`function Paused: HResult`	Извещает о том, что наступила пауза в распознавании (уведомление может быть вызвано методом `ISRCentral.Pausa`)

Таблица 3.18 *(окончание)*

Метод (Уведомление)	Описание
`function PhraseFinish (dwFlags: DWORD; qTimeStampBegin: QWORD; qTimeStampEnd: QWORD; pSRPhrase: PSRPhraseA; IUnknownResult: IUnknown): HResult; stdcall;`	Завершено распознавание фразы. Указатель `pSRPhrase` на структуру `SRPhrase`, через которую можно получить фразу. Последний параметр содержит указатель на интерфейс `IUnknown` объекта SpeechRecognition Result. Может включать несколько значений: `ISRNOTEFIN_RECOGNIZED` — фразу можно признать как достоверную; `ISRNOTEFIN_THISGRAMMAR` — текущая грамматика имела наибольший приоритет из всех активных грамматик
`function PhraseHypothesis (dwFlags: DWORD; qTimeStampBegin: QWORD; qTimeStampEnd: QWORD; pSRPhrase: PSRPhraseA; IUnknownResult: IUnknown): HResult; stdcall;`	Посылается приложению при подборе движком наилучшего варианта распознаваемой фразы. Обычно посылается несколько раз до уведомления `PhraseFinish`
`function PhraseStart (qTimeStampBegin: QWORD): HResult`	Посылается при старте распознавания фразы, после этого несколько раз может посылаться уведомление `PhraseHypothesis`

`function Training (dwTrain: DWORD): HResult`	Посылается, когда текущий движок требует тренировку диктора, грамматики или микрофона. Значение флага определяет, что именно требуется:
	`SRGNSTRAIN_MICROPHONE`
	`SRGNSTRAIN_GRAMMAR`
	`SRGNSTRAIN_GENERAL`

Самым главным для нас является уведомление `PhraseFinish`, поэтому здесь опишем лишь его реализацию (в демо-проекте описано большинство уведомлений).

```
function TSRGramNotifySink.PhraseFinish(dwFlags: DWORD; qTimeStampBegin:
QWORD; qTimeStampEnd: QWORD; pSRPhrase: PSRPhraseA; IUnknownResult:
IUnknown): HResult; stdcall;
var
 dwCnt: Integer;
 PSRW: PSRWord;
 g_szPhrase: String;
```

```
begin
  Result:=S_OK;
  try
    g_szPhrase := '';
    dwCnt := pSRPhrase.dwSize - SizeOf(pSRPhrase.dwSize);
    PSRW := @pSRPhrase.abWords;
    while dwCnt > 0 do
     begin
      g_szPhrase := g_szPhrase + PChar(@PSRW.szWord) + #32;
      Dec(dwCnt, PSRW.dwSize);
      Inc(PChar(PSRW), PSRW.dwSize)
     end;
    ShowMessage(g_szPhrase);
  except
  end;
end;
```

Смысл процедуры таков: параметр `pSRPhrase` является указателем на структуру `SRPHRASEA` (фраза), в ее поле `abWords` содержится список указателей на структуры типа `SRWORD` (одно слово), а в поле `dwSize` — ее размер.

```
SRPHRASEA = record
  dwSize:    DWORD;
  abWords : record end;
end;
TSRPhraseA = SRPHRASEA;
PSRPhraseA = ^TSRPhraseA;
```

Поэтому нам необходимо пользоваться дополнительным указателем `PSRWord`, чтобы по словам считать фразу из памяти.

```
SRWORDA = record
  dwSize:     DWORD;
  dwWordNum:  DWORD;
  szWord : record end;
end;
TSRWordA = SRWORDA;
PSRWordA = ^TSRWordA;
PSRWord = PSRWordA;
```

Рассмотрим процедуру выбора движков, она имеет много комментариев и, полагаю, в ней будет несложно разобраться. Код приведен в листинге 3.6.

Листинг 3.6. Обработчик события выбора модуля и загрузки в него грамматики

```
procedure TForm1.cbSREngineChange(Sender: TObject);
var
 NumFound: DWord;
 fsrModeInfo : SRModeInfo;{Для хранения информации о текущем движке}
 s: string;
 flnName: PWideChar;
begin
   try
     OleCheck(CoCreateInstance(CLSID_MMAudioSource, Nil, CLSCTX_ALL,
          IID_IAudioMultiMediaDevice, fIAMMS));
     if cbDevIn.ItemIndex=0 then
      OleCheck(fIAMMS.DeviceNumSet(WAVE_MAPPER))
     else
      OleCheck(fIAMMS.DeviceNumSet(cbDevIn.ItemIndex-1));

     CoCreateInstance(CLSID_SREnumerator, Nil, CLSCTX_ALL,
         IID_ISREnum, aSREnum);
     aSREnum.Reset;
     {Перескакиваем на нужный движок}
     Form1.aSREnum.skip(cbSREngine.ItemIndex);
     aSREnum.Next(1, fsrModeInfo, @NumFound);
     if assigned(sModeInfo) then
       {если fpModeInfo не равен nil}
       dispose(sModeInfo);
     new(sModeInfo);
     sModeInfo^:=fsrModeInfo;
     {загружаем движок}
     aSREnum.Select(sModeInfo.gModeID, fISRCentral, IUnknown(fIAMMS));

     //Создаем объект, компилирующий грамматику (GrammarCompiler)
     CoCreateInstance(CLSID_STGramComp, nil, CLSCTX_ALL,
       IID_ISTGramComp, fISTGramComp);
     //Указываем файл с грамматикой
     s:=ExtractFilePath(ParamStr(0))+'Grammar.txt';
     flnName:=StringToOleStr(s);
```

```
  //Загружаем ее в "компилятор"
  OleCheck(fISTGramComp.FromFile(flnName));
  GramErr := nil;
  //Компилируем грамматику
  OleCheck(fISTGramComp.Compile(GramErr, @Size));
  if Assigned(GramErr) then
    begin
     //Выводим сообщение
     ListBox1.Items.Add(WideString(GramErr));
     //Освобождаем память
     CoTaskMemFree(GramErr);
    end;

  {Создаем и регистрируем объект для получения уведомлений
   о состоянии распознавания речи}
  fISRNotifySink := TSRNotifySink.Create(Self);
  fISRCentral.Register(pointer(fISRNotifySink),
      IID_ISRNotifySink, p_dwKey);

  {Создаем объект, реализующий интерфейс получения
   уведомлений о распознанных фразах}
  fGramNotifySink:=TSRGramNotifySink.create(Self);
  {Загружаем грамматику в движок, с которым связан fISRCentral,
   также передаем указатель на интерфейс, который получает
   уведомления}
  OleCheck(fISTGramComp.GrammarLoad(fISRCentral,
     Pointer(fGramNotifySink), IID_ISRGramNotifySink, SRGramUnk));

  {Получив указатель на IUnknown объекта Grammar, запрашиваем
   интерфейс управления грамматикой ISRGramCommon}
  SRGramUnk.QueryInterface(IID_ISRGramCommon, fISRGramCommon);
  if Assigned(fISRGramCommon) then
    begin
      //Активизируем распознавание
      OleCheck(fISRGramCommon.Activate(Handle, False, nil));
      Act:=true;
      StatusBar1.Panels.Items[0].Text:='Слушаю...';
    end;
except
```

```
    MessageDlg('Ошибочка вышла! '+Exception(ExceptObject).ClassName+
     ' - '+Exception(ExceptObject).Message, mtError, [mbOk], 0);
  end;
end;
```

Вполне вероятно, что в списке движков будут присутствовать не все модули, установленные в системе, т. к. тут мы выводили модули, поддерживающие только контекстно-свободную грамматику.

Наверное, вы уже в нетерпении от желания поскорее узнать, что за таинственные правила описаны в файле грамматики Grammar.txt. Ничего особенного там нет. О формах граматики мы поговорим чуть позже, а пока удовлетворю ваше любопытство. Вот содержимое файла грамматики:

```
[Grammar]
LangID=1033
Type=cfg
[<Start>]
<Start> = Color
<Start> = Save file
<Start> = Open dialog
```

Здесь мы указываем тип грамматики (CFG, Context-free Grammar) и язык (1033, американский английский), а также распознаваемые фразы: "Color", "Save file" и "Open dialog". Запустите полный откомпилированный проект, который находится в каталоге 23_DirectSpeechRecognition, выберите необходимое устройство и движок (например, "English Continuous, Microphone (Microsoft)") и посмотрите на то, как распознается речь.

Полезно рассмотреть и другие методы, которые поддерживает интерфейс. Например, метод `ModeGet` возвращает указатель на структуру `SRModeInfo`, содержащую всю информацию о текущем используемом модуле распознавания речи. Распознавание речи, так же как и синтез, можно приостанавливать, и для этого служит метод `Pausa` центрального интерфейса. Отключить паузу можно методом `Resume` того же интерфейса. Если же мы деактивизируем грамматику, то полностью прекратим распознавание:

```
fISRGramCommon.Deactivate(nil);
```

Параметры речевых движков

Сейчас мы рассмотрим основные параметры речевых модулей. Все параметры хранятся в структуре `SRModeInfo`, которая весьма похожа на структуру

`TTSModeInfo` из DirectTextToSpeech API. Здесь также имеются название движка, идентификатор, фирма-производитель и т. д.

```
SRMODEINFOA = record
  gEngineID:                TGUID;
  szMfgName:                array [0..SRMI_NAMELEN-1] of AnsiChar;
  szProductName:            array [0..SRMI_NAMELEN-1] of AnsiChar;
  gModeID:                  TGUID;
  szModeName:               array [0..SRMI_NAMELEN-1] of AnsiChar;
  Language:                 TLanguageA;
  dwSequencing:             DWORD;
  dwMaxWordsVocab:          DWORD;
  dwMaxWordsState:          DWORD;
  dwGrammars:               DWORD;
  dwFeatures:               DWORD;
  dwInterfaces:             DWORD;
  dwEngineFeatures:         DWORD;
end;
```

Рассмотрим лишь имеющиеся отличия. Поле `dwSequencing` показывает поддерживаемую схему распознавания, оно может иметь одно из значений, приведенных в табл. 3.19.

Таблица 3.19. *Значения поля* `dwSequencing`

Константа	Значение	Описание
`SRSEQUENCE_DISCRETE`	0	Дискретная схема. Каждое слово в речи должно быть отделено паузой
`SRSEQUENCE_CONTINUOUS`	1	Слитная речь. Характерна для большинства движков
`SRSEQUENCE_WORDSPOT`	2	Распознавание отдельных слов. Слова можно произносить слитно, но движок распознает только некоторые из них
`SRSEQUENCE_CONTCFGDISCDICT`	3	Поддерживает контекстно-свободную грамматику и диктовку по словам

Параметр `dwMaxWordsVocab` показывает максимальное количество слов в активном словаре в любое время. Параметр `dwMaxWordsState` — максимальное количество слов в разделе грамматики.

Самое важное поле структуры — `dwGrammars`, каждый его бит показывает нам, какие типы грамматик поддерживает движок. Значения битов приведены в табл. 3.20.

Табилца 3.20. *Значения битов поля* `dwGrammars`

Константа	Бит	Грамматика
SRGRAM_CFG	0	Контекстно-свободная грамматика
SRGRAM_DICTATION	1	Грамматика диктовки
SRHDRTYPE_LIMITEDDOMAIN	2	Грамматика ограниченного домена

Биты поля `dwFeatures` символизируют поддержку особых возможностей движка, содержание которых приводится в табл. 3.21.

Табилца 3.21. *Значения битов поля* `dwFeatures`

Константа	Бит	Описание
SRFEATURE_INDEPSPEAKER	0	Движок не зависит от диктора и работает без обучения
SRFEATURE_INDEPMICROPHONE	1	Работа движка не зависит от микрофона, и движок может работать без перенастройки при его замене
SRFEATURE_TRAINWORD	2	Диктор может обучать движок на отдельных словах
SRFEATURE_TRAINPHONETIC	3	Диктор может тренировать движок на отдельных фонемах, которые используются при распознавании
SRFEATURE_WILDCARD	4	Движок поддерживает специальные метки wild-card в контекстно-свободной грамматике
SRFEATURE_ANYWORD	5	Движок может пытаться распознать любое слово
SRFEATURE_PCOPTIMIZED	6	Движок оптимизирован на работу через ПК, т. е. с более качественным голосом
SRFEATURE_PHONEOPTIMIZED	7	Движок оптимизирован на работу через телефон, т. е. на голос с качеством 8 кГц
SRFEATURE_GRAMLIST	8	Движок способен автоматически обновлять связи между грамматиками
SRFEATURE_GRAMLINK	9	Движок поддерживает списки имен
SRFEATURE_MULTILINGUAL	10	Движок способен распознавать речь сразу на нескольких языках

Табилца 3.21 *(окончание)*

Константа	Бит	Описание
SRFEATURE_GRAMRECURSIVE	11	Движок поддерживает рекурсии внутри граматики, но не может иметь рекурсию между грамматиками
SRFEATURE_SINGLEINSTANCE	13	Движок, как COM-объект, может быть загруженным только в одном экземпляре класса
SRFEATURE_THREADSAFE	14	Движок может быть вызван из любого потока
SRFEATURE_FIXEDAUDIO	15	Движок настроен на фиксированный источник звука, поэтому выбор устройства для него не нужен
SRFEATURE_SAPI4	17	Движок поддерживает все особенности SAPI 4.0

Проверку на подержку интерфейсов можно осуществить по полю `dwInterfaces`, имена ключевых констант начинаются с префикса `SRI_` и заканчиваются названием интерфейса. Приводить их здесь я не буду, при необходимости со всеми значениями можно ознакомиться в документации к SAPI 4.0.

В каталоге 24_SREngineViewer располагается своеобразный просмотрщик модулей распознавания речи, он отображает полную информацию о выбранном модуле.

Атрибуты распознавания речи

Как и в технологии Text-To-Speech, в Speech Recognition имеются атрибуты, но только уже распознавания речи. Под атрибутами здесь понимаются те параметры входной речи, на которые должен реагировать движок, обеспечивая более качественное ее распознавание. Например, входной уровень шума, эха, громкости речи и т. п.

Интерфейс для изменения атрибутов распознавания в прикладном интерфейсе DirectSpeechRecognition API — `ISRAttributes`. Получить его можно через метод `QueryInterface` у интерфейса `ISRCentral`. Интерфейс атрибутов имеет шесть парных методов, т. е. всего их двенадцать:

- `AutoGainEnableGet/Set` — методы управляют уровнем среза. "AutoGain" означает автоматическую корректировку мощности входного аудиопотока, чтобы при резких скачках его мощность оставалась примерно постоянной. При нуле эта опция отключается;

- `EchoGet/Set` — получает/изменяет (включает/отключает) учет поправки на эхо. Если распознавание проводится в большом помещении, где есть эхо, то следует включить эту опцию;
- `EnergyFloorGet/Set` — получает/изменяет пороговый уровень входного шума;
- `MicrophoneGet/Set` — получает/изменяет имя текущего микрофона для данного аудиоисточника;
- `RealTimeGet/Set` — получает/изменяет приоритет процесса распознавания речи;
- `SpeakerGet/Set` — получает/изменяет имя текущего диктора;
- `ThresholdGet/Set` — получает/изменяет пороговый уровень "уверенности" движка в распознавании фразы. Чем больше значения, тем больше должно быть "уверенности" у движка для того, чтобы признать какую-либо фразу распознанной;
- `TimeOutGet/Set` — в этих методах передаются сразу два параметра: `dwIncomplete` и `dwComplete`. Первый показывает, какое время ожидает движок, прежде чем отвергнуть неоконченную (оборвавшуюся) фразу. Например, во время произнесения определенной фразы пользователь может сделать паузу не длиннее `dwIncomplete`, иначе движок откажется ее распознавать. Второй параметр показывает, какое время ожидает движок, чтобы признать фразу завершенной. Значение параметра `dwIncomplete` должно всегда быть больше, чем у `dwComplete`.

Управление минимальными и максимальными значениями атрибутов, параметры которых имеют целочисленный тип (например, `WORD` и `DWORD`), производится так же как и управление атрибутами синтеза речи. Например, определение диапазона параметра `gain` производится следующим образом:

```
fISRAttr.AutoGainEnableSet(SRATTR_MINAUTOGAIN);
fISRAttr.AutoGainEnableGet(MinAutoGain);
lbGainL.Caption := IntToStr(MinAutoGain);
fISRAttr.AutoGainEnableSet(SRATTR_MAXAUTOGAIN);
fISRAttr.AutoGainEnableGet(MaxAutoGain);
lbGainH.Caption := IntToStr(MaxAutoGain);
```

В проекте каталога 25_DirectSR_Atr (рабочее окно приложения приведено на рис. 3.15) реализовано изменение всех основных атрибутов.

Кстати, в этом же проекте организовано отображение диалоговых окон движков распознавания речи. Уточню лишь, что для использования диалогов нужно вызывать методы интерфейса `ISRDialogs`.

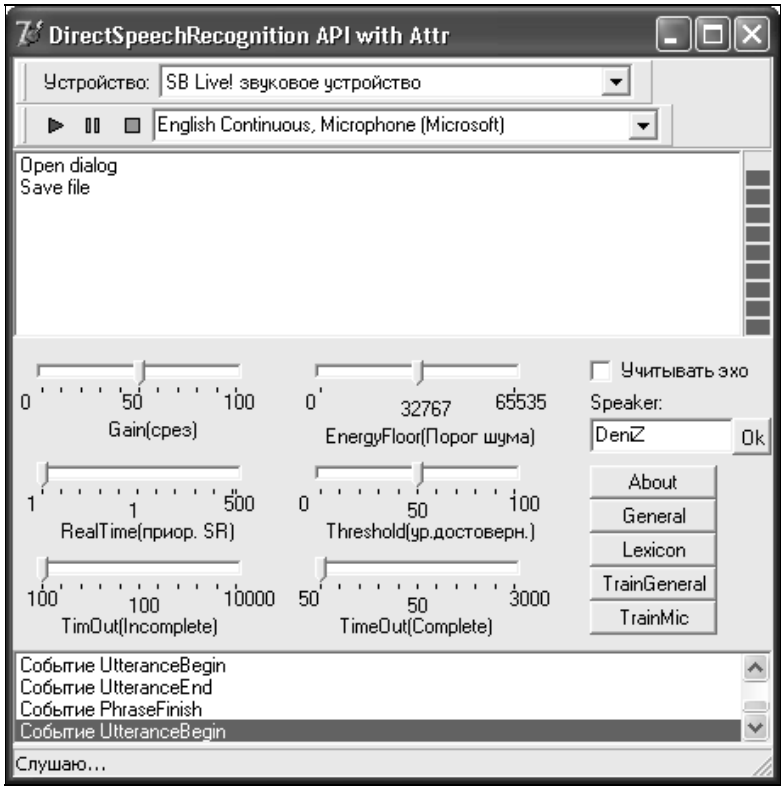


Рис. 3.15. Рабочее окно приложения с поддержкой атрибутов

Подготовка грамматик

Перед распознаванием желаемых фраз нужно подготовить файл грамматики, который будет содержать правила для движка. Благодаря этому облегчается функционирование его алгоритмов и повышается качество распознавания.

Структура грамматического файла похожа на структуру ini-файла. В каждом таком файле должна быть секция `Grammar`, где указываются настройки грамматики: ее язык и тип.

```
[Grammar]
LangID=1033
Type=cfg
```

Тип может иметь значения `CFG` (context-free), `dictation` и `limited-domain`. Указывать тип грамматики необязательно, по умолчанию она имеет тип `CFG`.

Другие необязательные параметры — `Dialect` и `CopmpileTo`. Первый обозначает диалект языка (по умолчанию будет пустая строка), а второй — тип грамматики, в который должен компилироваться исходный текст грамматических правил (по умолчанию равен значению параметра `Type`).

В грамматическом файле должна присутствовать секция `<start>`, с которой начинается ввод допустимых фраз и выражений. Эта секция обязана включать хотя бы одну фразу с одноименным параметром.

```
[<Start>]
<Start> = Open dialog
```

С данной секции начинается составление шаблонов. Такие секции называются *правилами* (rules) и их названия пишутся внутри угловых скобок для того, чтобы отделить их от других типов списков, которые не экспортируются подобно правилам (они называются *импортируемыми списками*). В секции `<start>` должны присутствовать все основные выражения, которые поддерживает программа. Каждая фраза может содержать *списки имен*, т. е. набор выражений, который находится в этом месте (например, "Хочу посмотреть (OPT, PTP, и т. д.)"). В таких случаях создают секцию с любым именем и определяют в ней выбор, а в главную фразу вставляют название секции, из которой берутся выражения. Например, пусть нам не хочется прописывать три команды: "New file", "Open file" и "Save file". Тогда можно поступить так:

```
[<Start>]
<Start> = <cmd> file
[<cmd>]
<cmd> = New
<cmd> = Open
<cmd> = Save
```

Существует несколько стандартных наборов правил, которые могут понимать движки, в частности, правила: формата номеров (`<PhoneNumber>`), годов (`<Year>`), месяцев (`<Month>`), даты (`<Date>`), времени (`<Time>`), чисел (`<Digits>`), дробей (`<Float>`), отдельных цифр (`<0..9>`) и т. д.

В тех случаях, когда пользователю не обязательно проговаривать какую-либо часть фразы, перед ней вставляется команда `[opt]` (от слова *optional*, необязательный). Например:

```
[<Start>]
<Start> = <cmd> [opt] file
[<cmd>]
<cmd> = New
<cmd> = Open
<cmd> = Save
```

С такой грамматикой движок способен распознавать как фразы "New file", "Open file" и "Save file", так и просто команды "New", "Open" или "Save".

Правила, которые пишутся в угловых скобках, экспортируются движку и поэтому называются *экспортируемыми правилами* (exporting rule). Если нужно создать список, элементы которого не могут распознаваться отдельно, а только в составе какой-либо фразы, то в названии секции списка не должно быть угловых скобок, а фразы должны перечисляться без указания так называемого параметра (если ориентироваться на термин, принятый для ini-файлов). Такие списки называются *импортируемыми* (Importing list), и все они должны быть описаны в разделе `Lists`:

```
[Grammar]
LangID=1033
Type=cfg

[Lists]
=what

[<Start>]
<Start> = <cmd> [opt] <what>

[<cmd>]
<cmd> = New
<cmd> = Open
<cmd> = Save

[what]
=file
=dialog
```

Используя данную грамматику, движок может распознать команды "New", "Open", "Save", как отдельно, так и в сочетании со словами "file" или "dialog". Однако по-отдельности слова "file" и "dialog" распознаны быть не могут.

Существует много других грамматических правил, но описанные являются основными, и благодаря им можно построить необходимые шаблоны фраз. Файлы с описанными здесь примерами находятся в каталоге 26_DirectSR_Gramm, в нем также есть другие интересные примеры грамматик из Speech API SDK. Кроме того, в том же каталоге находится программа, через которую удобно загружать различные грамматики из файлов, расположенных в текущей директории. За большим количеством примеров можно обратиться к документации Speech API SDK.

Элемент управления DirectSR

Управление распознаванием речи в компоненте DirectSpeechRecognition через элемент ActiveX выполняется так же просто, как и в случае со всеми уже рассмотренными нами интерфейсами.

Элмент управления, предоставляющий доступ к DirectSpeechRecognition, называется DirectSR. Он устанавливается обычным образом, как и все элементы ActiveX. В окне установки элемента управления в списке нужно выбрать пункт "Microsoft Speech Recognition" или, если такого пунтка нет, выбрать в каталоге Speech директории системы файл Xlisten.dll.

После установки к "уху от VoiceCommand" (если вы устанавливали этот элемент ActiveX) добавится точно такое же "ухо от DirectSR". Для работы с элементом управления разместите компонент на форме и проинициализируйте свойство `Initialized`, т. е. присвойте ему значение 1. Используйте метод `GrammarFromFile`, чтобы загрузить грамматику из файла, и метод `Activate`, чтобы активизировать ее.

Создать список движков можно, добавив в процедуру обработки события создания формы следующий код:

```
procedure TForm1.FormCreate(Sender: TObject);
var
 i: integer;
begin
 for i:=1 to DirectSR1.CountEngines do
  cbEngine.Items.Add(DirectSR1.ModeName(i));
end;
```

Применение выбранного движка также осуществляется очень тривиально:

```
procedure TForm1.cbEngineChange(Sender: TObject);
begin
 DirectSR1.Select(cbEngine.ItemIndex)
end;
```

Чтобы получить формулировку распознанной фразы, нужно обработать событие `OnPhraseFinish`, например:

```
procedure TForm1.DirectSR1PhraseFinish(Sender: TObject; flags, beginhi,
  beginlo, endhi, endlo: Integer; const Phrase, parsed: WideString;
  results: Integer);
begin
 ShowMessage(Phrase);
end;
```

Полный проект смотрите в каталоге 27_DirectSrAX. Думаю, вы уловили суть, т. к. принцип одинаков во всех элементах управления Speech API.

И все-таки рекомендую пользоваться COM-объектами вместо аналогичных активных компонентов, а то слишком уж все просто обстоит с этими ActiveX (**;-)**). Хотя, разумеется, это личное дело каждого, и все зависит от конкретной ситуации.

3.8. Резюме

Все речевые технологии можно разделить на две большие категории по критерию направленности речи: синтез речи (направлена к нам) и распознавание речи (направлена в компьютер). В распознавании речи выделяют технологии распознавания команд и технологии диктовки произвольного текста.

Одним из самых известных прикладных интерфейсов программирования для реализации речевых технологий обладает пакет функций Speech API (SAPI). В его основе лежит общий набор системных функций, доступных для использования приложением. Эти функции сами по себе не синтезируют и не распознают речь, они управляют другими модулями, которые это осуществляют.

В SAPI выделяются следующие технологии работы с речью: Command and Control, Dictation, Text to speech, Telephony. Поэтому весь интерфейс программирования речи разбивается на несколько: Voice Command API, Voice Dictation API, Voice Text API, Voice Telephony API, DirectSpeechRecognition API, DirectTextToSpeech API.

Первые три интерфейса считаются интерфейсами высокого уровня (как следствие, они чуть более медленные), т. к. они скрывают некоторые тонкости управления речевыми функциями, а последнии два — интерфейсами низкого уровня (значит, в чем-то они и более быстрые). Хотя для Voice Telephony API не выделено место в базовой схеме SAPI4, он также считается интерфейсом высокого уровня.

Для синтеза и распознавания речи Speech-функции обращаются к соответствующим модулям (движкам, engine, речевым синтезаторам, TTS-генераторам). Эти модули оформлены в виде отдельных библиотек и могут разрабатываться кем угодно. Благодаря этому различные фирмы-разработчики создают собственные модули синтеза (TTS-Engine, TTS-генераторы, синтезаторы) и распознавания речи (SR-Engine), причем первых пока больше.

Технология Speech API построена на основе модели COM, соответственно и работа со всеми ее интерфейсами заключается в обращении к определенным COM-объектам.

В Speech API существуют не только объекты, соединяющие нас с какими-либо речевыми модулями, но и объекты, предоставляющие доступ к аудиоустройствам вывода звука и к его источникам. Такие объекты называются Audio Objects.

Движок синтеза или распознавания речи представляет собой набор динамических библиотек и может содержать в себе несколько голосов (для TTS) или распознавателей (для SR), идентифицирующихся в системе по глобальному уникальному идентификатору (GUID). Это свойство конкретного синтезатора фигурирует в его параметрах как `ModeID`. Кроме того, сами движки имеют другой независимый идентификатор — `EngineID`.

Схема использования функций синтеза речи через низкоуровневый интерфейс такова. Сначала необходимо выбрать устройство вывода звука через специальные методы интерфейсов определенных COM-объектов для связи с аудиоустройствами. Это может быть как звуковая карта, так и аудиофайл. Затем нужно выбрать речевой модуль, а точнее — голос, через который будет осуществляться синтез, после чего через дополнительные интерфейсы объявить, что данный модуль будет использоваться для синтеза речи. И наконец, следует подать буфер текста для воспроизведения.

Для распознавания речи используется набор правил (грамматики), через которые модулю распознавания сообщается круг фраз и выражений, которые должны распознаваться. Грамматические правила нужно составлять либо в определенном формате, понятном движку, чтобы непосредственно загружать их в него, либо записывать их в отдельном файле и использовать специальный объект Grammar Compiler для преобразования данных грамматики в нужный формат, а уже затем загружать в модуль.

Для использования низкоуровневого распознавания речи нужно также сначала определить источник звука, а потом, выбрав модуль распознавания речи, загрузить грамматику для данного движка и активизировать процесс распознавания.

Глава 4

"Примочки" к MS Agent

Настоящая глава книги посвящена различным дополнительным компонентам и программным модулям, созданным для Microsoft Agent и технологии Speech API. В основном это разработки компании Microsoft, однако другие разработчики также принимают участие в создании различных так называемых "примочек" к MS Agent. В частности, интересна технология Balloon-Dialog, суть которой заключается в расширении функциональности стандартной выноски персонажа — воздушного шарика (Balloon).

Естественно, здесь мы поговорим о собственноручном создании персонажей, о том, как выпустить на свет маленького анимированного человечка. Узнав о том, как подготавливаются LWV-файлы в лингвистическом редакторе вы научитесь создавать специальные звуковые файлы с дополнительной лингвистической информацией, которая поможет агенту правильно открывать рот во время воспроизведения речи. Напоследок я расскажу о компоненте BalloonDialog v6.5, а сначала мы рассмотрим "примочку" под названием Microsoft Speech Control Panel.

4.1. Апплет Microsoft Speech Control Panel

По названию кое-кто может уже догадаться, что речь пойдет о некоем апплете Панели управления. Это действительно так. В предыдущей главе мы много раз представляли интерфейс в своих приложениях для обзора и выбора имеющихся движков синтеза и распознавания речи. Понятно, что неудобно запускать свои собственные приложения для просмотра допустимых речевых модулей (хотя, конечно, свое всегда приятнее), в этих целях создан специальный апплет Microsoft Speech Control Panel. Он умеет наглядно отображать установленные речевые модули в виде древовидной структуры, упорядоченной по фирмам-изготовителям, языкам и голосам.

В системе Windows 98 после установки пакета функций Speech API и апплета Microsoft Speech Control Panel в окне **Панель управления** появится пункт **Speech** (Речь), как показано на рис. 4.1.

Рис. 4.1. Окно **Панель управления** с пунктом **Speech** речевого апплета

Выбор этого пункта будет приводить к появлению диалогового окна **Speech Properties** (Свойства речи) (рис. 4.2). Как видите, у меня на компьютере установлены речевые синтезаторы от разных фирм, а также для различных языков.

В операционной системе Windows XP (и в более "свежих") по умолчанию установлен пакет функций Speech API v5.1, который несколько отличается от того же пакета версии 4.0, однако обе версии хорошо уживаются вместе, так же как и Microsoft Agent v1.5 с Microsoft Agent v2.0. Windows XP, если вы успели заметить, имеет в меню Панели управления свой пункт **Речь** (рис. 4.3), поэтому при установке SAPI 4 новый пункт не появляется, зато в диалоговое окно **Свойства речи** (Пуск | Панель управления | Речь) добавляется вкладка **Прочие** (рис. 4.4) с сообщением о том, что на вашем компьютере установлен также и SAPI 4.

Если в диалоговом окне **Свойства речи** щелкнуть на кнопке **Панель управления SAPI4**, откроется диалоговое окно **Speech Properties** апплета Microsoft Speech Control Panel. На рис. 4.4 показаны оба этих окна.

На вкладке **Speech** диалогового окна **Speech Properties** в группе элементов **Installed speech engines** (Установленные речевые движки) отображается древовидная структура речевых модулей. Первый уровень ветвления названий речевых движков в иерархической структуре (в "дереве движков") (см. рис. 4.4) — по критерию ввода/вывода (т. е. движки делятся на модули

синтеза речи (речевые синтезаторы) и модули распознавания речи), затем — по фирме-изготовителю. Каждый узел фирмы содержит дочерние узлы языков модулей, и для каждого языка выводится список голосов или модулей распознавания речи.

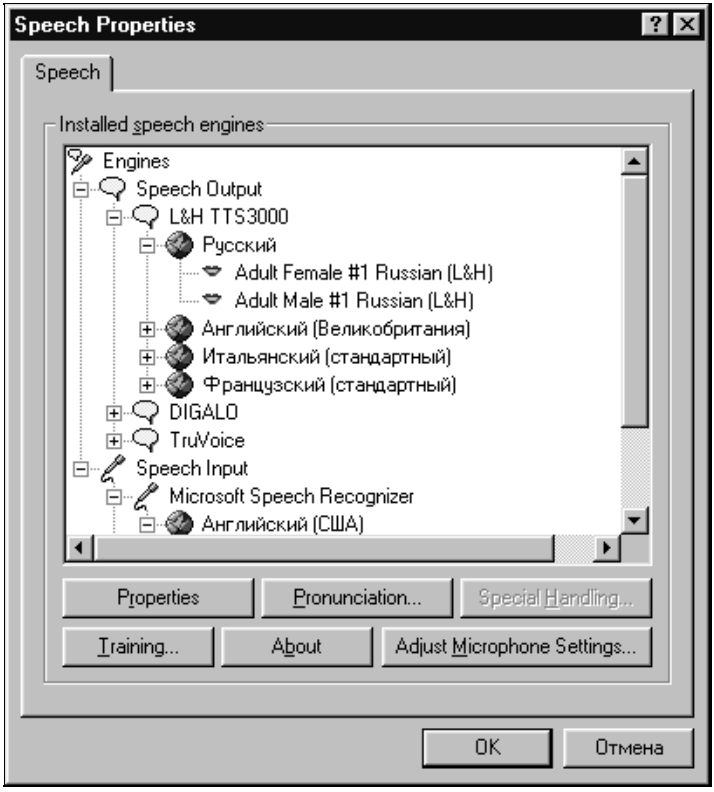


Рис. 4.2. Диалоговое окно **Speech Properties**

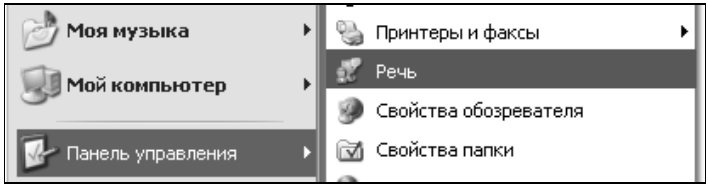


Рис. 4.3. Пункт **Речь**
меню Панели управления в Windows XP

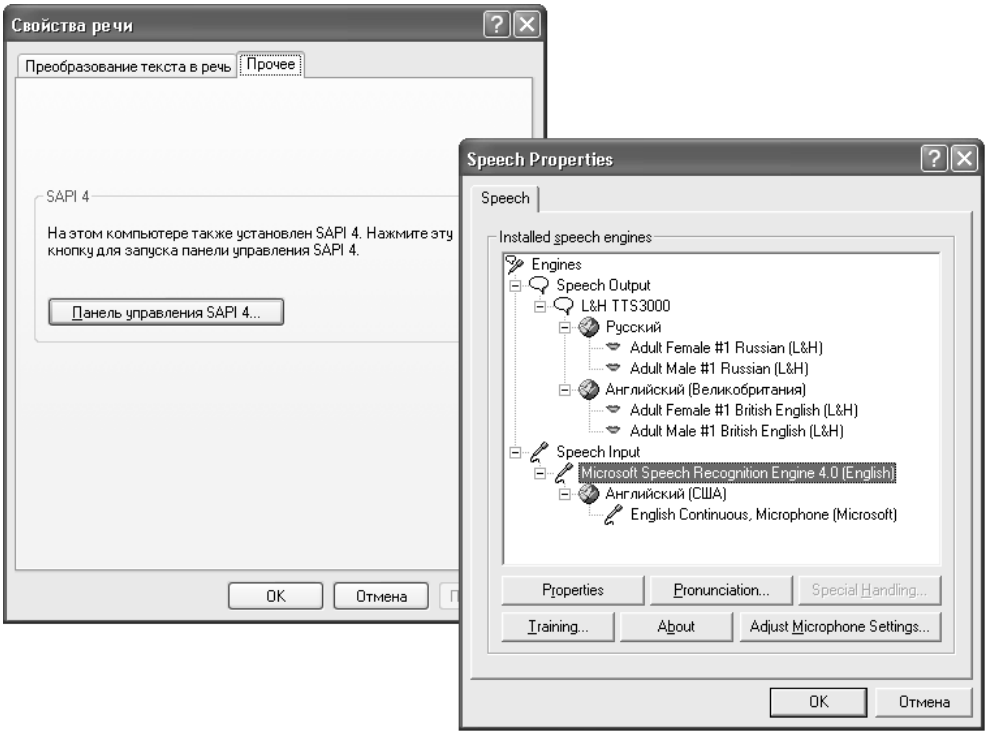


Рис. 4.4. Диалоговое окно **Свойства речи** в Windows XP и диалоговое окно **Speech Properties** от SAPI 4.0

При выделении фирмы-изготовителя в нижней части диалогового окна под деревом движков становятся доступными некоторые кнопки. Для речевых синтезаторов это могут быть кнопки **Properties** (Свойства), **Pronunciation** (Произношение) или **About** (О модуле). В зависимости от того, какие диалоги поддерживает модуль, те или иные кнопки могут быть подсвеченными. Например, движки от фирмы Digalo могут похвастаться только информацией о модуле, в то время как модули Lernout & Hauspie, кроме окна **About**, имеют как настраиваемый словарь произношений, так и окно свойств синтеза речи.

Для модулей распознавания речи активизируются кнопки **Properties** (Свойства), **Training** (Тренировка) (служит для вызова режима тренировки модуля на распознавание речи), и конечно, кнопки **About** (О модуле) и **Adjust Microphone Settings** (Корректирующие настройки микрофона).

Думаю, вы оценили удобство наглядности представления информации о речевых модулях, установленных в вашей системе с помощью апплета Панели управления Microsoft Speech Control Panel.

4.2. Редактор персонажей Microsoft Agent Character Editor

Настало время научиться создавать свои собственные персонажи, и в этом нам поможет специальный редактор "от Microsoft". Только в нем возможно собирать (иначе говорят "компилировать", как о программе) персонажи из отдельных наборов картинок. У многих читателей, скорее всего, уже есть любимый агент, однако не менее интересно создать собственный персонаж, воспользовавшись специальной утилитой Microsoft Agent Character Editor. Ссылка, по которой ее можно скачать с сайта Microsoft, приведена в *приложении 1*.

Я предполагаю, что среди тех, кто держит сейчас в руках эту книгу, немного найдется профессиональных аниматоров, работающих с пакетом трехмерной графики 3D Studio Max так же свободно, как, например, программисты — в Блокноте. Поэтому мы не будем касаться здесь процесса создания "крутых" агентов. Для нас важнее овладеть технологией создания анимированных персонажей, а не самой анимацией в прикладных пакетах. Не расстраивайтесь, если вы не в состоянии создать высококачественную анимацию в специальных пакетах, ведь даже двухмерные персонажи, выполненные "в надлежащем виде", могут составить достойную конкуренцию их трехмерным собратьям.

Перед тем как начать создавать свой персонаж, давайте рассмотрим основные этапы его разработки. Так как роль анимированного персонажа — в том, чтобы укрепить социальный контакт между пользователем и компьютером, то, следовательно, его создателю нужно хорошо знать и понимать принципы поведения живых систем в реальном мире.

Замечание

В примерах данной главы используется исключительно просто нарисованный персонаж (синий человечек), подобного которому легко сможет подготовить любой начинающий программист.

Практические советы по анимированию

Приступая к работе над собственным персонажем, следует помнить, что нередко вам будет сложно быстро воплотить ваш замысел в жизнь. Вначале нужно просто почувствовать, что собой представляет этот процесс. Поэтому, создавая наш первый персонаж, мы не будем уделять большого внимания основным рекомендациям, приведенным чуть ниже. Однако если вы станете всерьез задумываться о создании качественного персонажа, то рано или поздно придете к аналогичным выводам (но все же нелишним будет хотя бы бегло их просмотреть).

Итак, занимаясь анимированием персонажа, необходимо помнить о следующих ключевых моментах анимации.

- Изменение формы и размеров тела персонажа (деформация) — продумайте степень деформации персонажа. Так как во время анимации тело изменяет свою форму, удлиняется или укорачивается, это дает возможность пользователю понять природу персонажа.
- Упреждение — эта анимация происходит непосредственно перед действием персонажа. На принципе упреждающего действия основан весь реальный мир. Например, движение существа в одном направлении часто начинается с движения в противоположном направлении: перед выдохом мы делаем вдох, перед тем как шагнуть вперед отклоняемся назад. Без подобных упреждающих действий поведение персонажа будет резким, механическим, неестественным.
- Продолжительность или быстрота действий — также определяют природу персонажа. Эти качества помогают пользователю представить себе размер и вес персонажа. Массивный персонаж не может двигаться быстро, в то время как небольшие существа предполагают соответствующую скорость движения ("юркость"), что и помогает им обратить на себя внимание.
- Организация фона и опоры персонажа — включает такие элементы, как световые эффекты и углы обзора персонажа. Эффективная организация также предполагает определенную направленность глаз ("взгляда") персонажа на тот объект, к которому вы хотите обратиться.
- Переходы и перекрытие действий — важно учитывать эти особенности, т. к. реальные живые существа никогда не двигаются внезапно и резко. Также заметьте, что, к примеру, человек, двигая рукой, совершает определенные движения кистью руки и пальцами. Все действия начинаются и завершаются, а также плавно переходят одни в другие.
- Дугообразные пути движения — являются особенностью поведения организмов в реальном мире. Чем медленнее движение, тем более причудливы пути движения различных частей тела. Наоборот, при быстром движении пути складываются в более прямолинейные траектории.
- Преувеличение — отдельные части образа персонажа можно использовать для подчеркивания его природы. Обычно преувеличивают форму, размер, цвет, эмоции и действие персонажа. Однако пользоваться надо этим с осторожностью, т. к. слишком сильное преувеличение может быть неправильно расценено пользователем.
- Вторичные действия — т. е. те действия, которые не должны преобладать над первичными, — должны быть также определены для персонажа. Обычно первичными действиями считаются движения тела, а вторичны-

ми — изменение выражения лица, т. к. последнее менее заметно на общем плане поведения персонажа, что не умаляет его значимости.

- Твердость рисунка — не самый лучший стиль создания ряда изображений. Не следует делать симметричные движения персонажа зеркальными. Посмотрите, например, "зеркально" ли персонаж Peddy (Попугай) показывает крылом налево и направо? Действия симметричные, но реализуются они не зеркально, хотя некоторый баланс между одинаковыми действиями существует. Благодаря этому персонаж еще больше приближается к реальным существам.
- Наконец, изучите потенциальную аудиторию клиентов персонажа — выберите для него индивидуальность. Он необязательно должен быть фотореалистичным. Простыми формами тоже можно добиться передачи почти всей необходимой эмоциональной окраски существа.
- Помните об оптимальной продолжительности анимации — она не должна быть длинной. Обычно 5—7 секунд — более чем достаточное время для самого продолжительного действия, а для быстрых анимаций типа GestureAt оно составляет около 0,5—1,5 секунд.

Конечно же, существует множество других тонкостей в разработке анимированного помощника, однако мы остановимся на этих основных и перейдем непосредственно к его созданию.

Основные понятия и принципы

Как и любая традиционная анимация, мультипликация персонажей MS Agent складывается из набора слегка различающихся картинок. При их последовательном воспроизведении плавные изменения на изображениях создают иллюзию действий. Картинки следует подготавливать любым доступным вам способом. Специальный редактор Microsoft Agent Character Editor занимается всего лишь сборкой персонажей из отдельных картинок.

Файл персонажа имеет определенную структуру и включает информацию о персонаже, необходимые настройки речи, выноски (Balloon), а также самое важное — наборы анимаций и связи, определяющие, какие анимации ассоциированы с определенными состояниями. Хотя форматы персонажей, как мы уже знаем, бывают разными, содержат они всегда одну и ту же информацию.

Каждая анимация из комплекта персонажа состоит из набора фреймов, или кадров (frames, кадр), каждый кадр может включать в себя один или более снимков (images, образ). Изображения могут быть двух форматов — Windows BMP и GIF. При этом нужно следить за тем, чтобы картинки использовали палитру в 256 цветов, вместе с цветом прозрачности. Это означает, что цветовая палитра вашего персонажа может использовать до 255 цветов. Возможно, такие тонкости излишни, ведь в ходе использования графиче-

ских редакторов (например, Adobe Photoshop) проблема решается сама собой. Кому-то такое "малое" (по сегодняшним меркам) качество цвета покажется слабым, но это ограничение можно понять, ведь чем больше цветов, тем больше вероятность, что на каких-либо компьютерах цветовая палитра будет отображена неправильно. Тем более что при увеличении количества цветов резко возрастает размер конечного файла персонажа.

Размер персонажа определяется величиной его кадров. Оптимальный размер кадра — 128×128 пикселов. В принципе, это не жесткое ограничение, и агент может быть практически любого размера, однако в случае большего размера так же увеличивается и конечный размер файла. При выборе образа персонажа избегайте большого количества маленьких пустых мест (прозрачного цвета) в середине его изображения. Персонажи выводятся посредством регионов[1], поэтому наличие замысловатых областей с пробелами внутри сказывается на быстроте воспроизведения анимации.

Использование такого красивого графического эффекта, как сглаживание, может отразиться на качестве воспроизведения персонажа. Например, размытие границ изображения отрицательно повлияет на отображение персонажа на любом другом фоне. В таком случае будет наблюдаться своеобразная шероховатость, и грязь будет близка к цвету фона. Поэтому постарайтесь, чтобы границы персонажа в кадрах были четкими, хотя это не исключает использование размытия внутри самого образа персонажа.

Создавая последовательность кадров мультипликаций, найдите во всех действиях персонажа нейтральную позицию, на которой анимация должна заканчиваться. Как правило, это удобно для реализации переходов между разными анимациями. Если все они заканчиваются в одном положении, то перетекание между действиями будет плавным. Разумеется, это не догма, и можно поступать по-другому. Например, все анимации начинать с нейтральной позиции, а заканчивать в любом удобном положении. Правда, затем в ходе создания персонажа, кроме анимаций, предназначенных для воспроизведения, могут быть созданы мультипликации, назначенные как возвратные анимации (Return animation) для определенного состояния.

Полагаю, все эта теория вам уже поднадоела, и пора перейти непосредственно к практике.

Начинаем создавать свой персонаж

Прежде чем приступить к разработке собственного персонажа, следует установить специальный редактор. По умолчанию при установке он "просится" в X:\Program Files\Microsoft Agent\Tools\, причем копирует туда один файл:

[1] Прямоугольная форма окна персонажа вырезается по прозрачному цвету, благодаря чему создается регион вывода.

ace.exe. Это и есть редактор персонажей. Вопреки названию "Editor", данная программа не может редактировать уже собранные (откомпилированные) персонажи, она способна лишь редактировать проект персонажа, т. е. в ходе создания персонажа можно сохранять в файле промежуточные результаты во внутреннем формате программы, с тем чтобы продолжить его создание позже.

Интерфейс программы необычайно прост, но все же при первом взгляде на него могут возникнуть некоторые вопросы, т. к. с некоторыми понятиями, используемыми в этом редакторе, мы пока незнакомы. Посмотрим на главное окно программы (рис. 4.5).

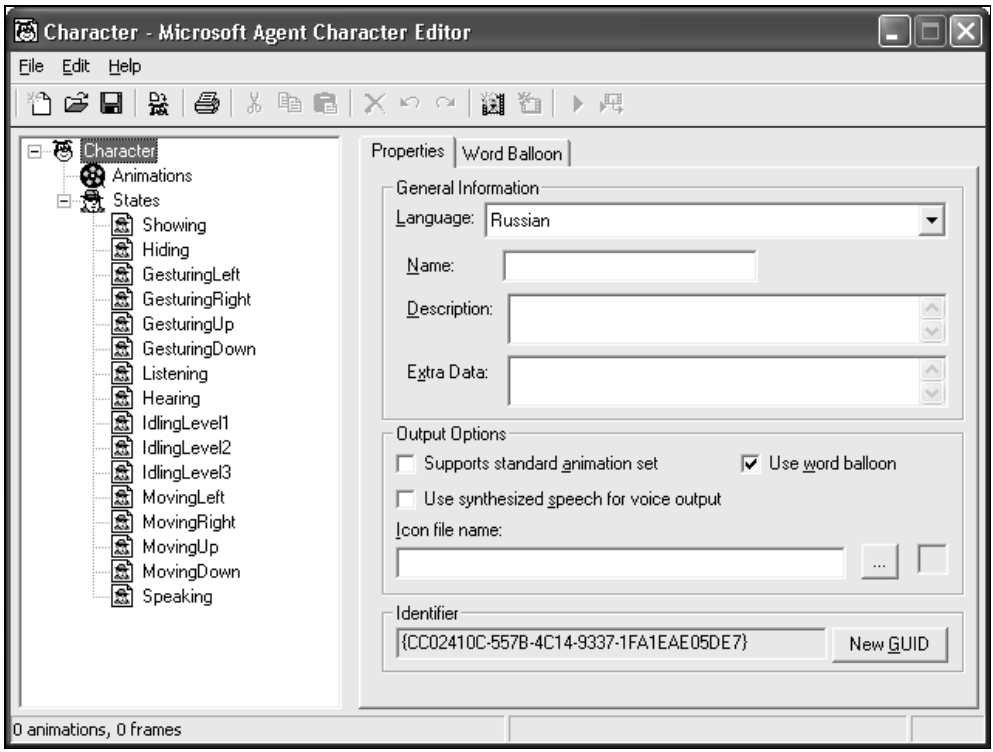


Рис. 4.5. Главное окно программы Microsoft Agent Character Editor

Изначально нашему вниманию представлена вкладка **Properties** (Свойства), с помощью элементов которой закладываются основные свойства персонажа, как ранее мы их называли — "личные", т. к. они характеризуют персонаж как виртуальную анимированную личность. Это свойства, задаваемые в полях ввода **Name** (Имя), **Description** (Описание), **Extra Data** (Дополнитель-

ные данные), значения которых доступны через одноименные свойства объекта `Character`.

Начиная разговор об этих свойствах, я уже упоминал, что имя персонажа отображается в подсказке значка системного лотка на текущем языке системы, да и все остальные свойства персонажей от Microsoft, как вы, наверное, заметили, также отображаются в окнах свойств по-русски. Понятно, что персонаж не осуществляет перевод, так что разработчик должен сам определить, как будет звучать на разных языках имя "анимированной живности". Для этого существует раскрывающийся список **Language** (Язык), в котором, выбирая нужный язык, вы определяете, как будут отображаться и звучать все личные строковые свойства персонажа.

Поле имени персонажа должно содержать не более 32 символов, описание — не более 255, таково же ограничение и на поле дополнительных данных (**Extra Data**), которое может содержать любую информацию. Если же вы планируете использовать создаваемого помощника в MS Office, придерживаясь в нем должной структуры, то из свойства дополнительных данных будут черпаться фразы определенного типа. Полное содержание поля дополнительных данных таково:

```
IntroPhrase1~~IntroPhrase2~~IntroPhrase3^^ReminderPhrase1~~ReminderPhrase
2~~ReminderPhrase3
```

Параметры: `IntroPhrase` — синонимический ряд приветствий, который отображается в галерее персонажей Office; `ReminderPhrase` — фразы напоминания (они используются для отображения в выноске Balloon при напоминаниях через Outlook).

Под рассмотренной группой элементов управления **General Information** (Общая информация) (см. рис. 4.5) следует группа элементов **Output Options** (Параметры вывода), в ней как раз и указываются настройки персонажа, поддержка основных способов вывода информации. Как известно, персонаж может отображать информацию через воспроизводимую анимацию, с помощью "шаровоздушной" выноски Balloon, а также через речь. Три флажка: **Supports standard animation set** (Поддержка стандартного набора анимаций), **Use synthesized speech for voice output** (Использовать синхронизацию речи и голосового вывода) и **Use word balloon** (Использовать текстовую выноску) регулируют эти возможности.

- **Supports standard animation set** — включает поддержку стандартного набора анимаций для персонажа, т. е. редактор при компоновке персонажа будет проверять наличие всех стандартных мультипликаций (см. документацию MS Agent SDK, раздел *The Office Animation Set*). Как раз это и даст нам отображение версии набора стандартных анимаций в окне свойств файла персонажа и в свойстве `Version` объекта `Character`. Если поддержка не включена, то будет отображаться слово "Нет".

- **Use synthesized speech for voice output** — устанавливает поддержку речевого вывода персонажа. При установке этого флажка добавится еще одна вкладка **Voice** (Голос), на которой конфигурируются дополнительные речевые параметры. По умолчанию флажок не установлен.
- **Use word balloon** — подключает использование текстовой выноски Balloon. По умолчанию флажок включен, поэтому в главном окне программы кроме вкладки **Properties** (Свойства) присутствует вкладка **Word Balloon** (Текстовая выноска) с элементами настройки "воздушного шарика" персонажа.

Чуть ниже этих флажков располагается поле ввода, где следует вводить имя файла пиктограммы персонажа, которая будет отображаться в системном лотке (SysTray). Выбрать ее можно через стандартное окно выбора файлов после щелчка на кнопке с троеточием рядом с полем ввода. После выбора пиктограмма тут же появится в крайнем окошечке справа от кнопки.

Под группой **Output Options** находится последняя группа, состоящая из одного элемента — это поле **Identifier** (Идентификатор), глобальный уникальный идентификатор GUID. Рядом с полем идентификатора в поле только для чтения расположена кнопка **New GUID** (Новый GUID), которая генерирует новые значения GUID. Если текущий номер вам по каким-либо причинам не подходит, например, вы создаете новый персонаж из предыдущего, то идентификатор потребуется обновить.

Как только вы заполните поля страницы свойств персонажа, можно будет приступить к настройке выбранных опций. Рассмотрим опции выноски Balloon. Для этого перейдите на вкладку **Word Balloon** (см. рис. 4.6). В верхней группе элементов **Default Display Settings** (Настройки отображения по умолчанию) настраивается количество символов в строке, количество строк в шарике. Установка флажка **Auto hide** (Автоматическое скрытие) обеспечивает автоматическое скрытие Balloon после использования, а с помощью флажка **Auto pace** (Автоматический темп) задается отображение слов синхронно с чтением (если флажок снят, то слова будут появляться мгновенно).

В группе элементов **Sample** (Пример), кроме примера выноски, присутствуют кнопки, с помощью которых задаются шрифт и цвета различных частей шарика, таких как фон, бордюр и цвет самого текста.

Если вы включили поддержку речи на вкладке **Properties**, то в окне программы должна быть доступна вкладка **Voice** (Голос) (рис. 4.7).

На этой вкладке определяют идентификатор речевого синтезатора, используемый по умолчанию, а также скорость и тон речи. Причем на выбор предоставляется набор движков, установленных в системе. Здесь же можно проверить введенные параметры на реальном примере, т. е. ввести произвольный текст в поле редактирования, расположенное в нижней части вкладки, и нажать кнопку **Test** (Проверить). После того как сконфигурированы все основные параметры персонажа, можно переходить непосредственно к созданию анимаций.

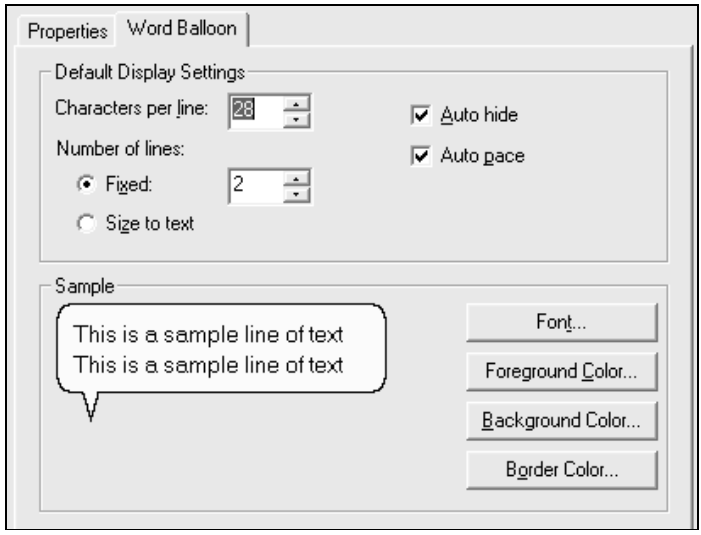


Рис. 4.6. Вкладка **Word Balloon** главного окна программы Microsoft Agent Character Editor

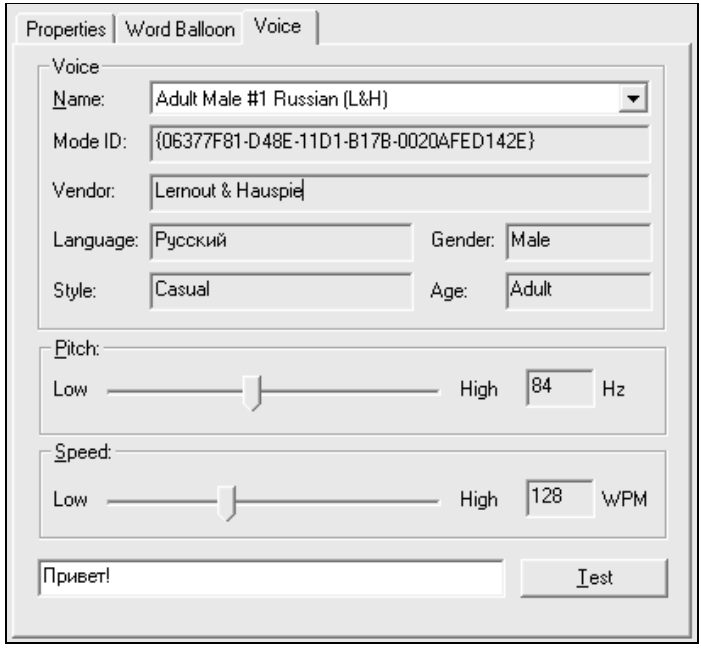


Рис. 4.7. Вкладка **Voice** главного окна программы Microsoft Agent Character Editor для настройки голосового вывода

Подготовка изображений

Анимированный персонаж можно подготовить любым доступным способом. Например, в программе 3D Studio Max, или в специальной программе, предназначенной для создания человекоподобных существ: CharacterStudio. Главное, чтобы в результате вашего труда появился набор картинок формата BMP или GIF. Если они получены простым сохранением кадров или через использование клавиши <Print Screen>, то скорее всего потребуется их дополнительная обработка. Для того чтобы картинки хорошо отображались в редакторе персонажей, а затем и при работе, их палитра не должна использовать более 256 цветов, с учетом прозрачного цвета. Необходимую схему действий рассмотрим на примере работы с программой Adobe Photoshop. Почти во всех версиях принципиальных отличий не будет. Итак, откройте один из кадров в программе Adobe Photoshop (рис. 4.8).

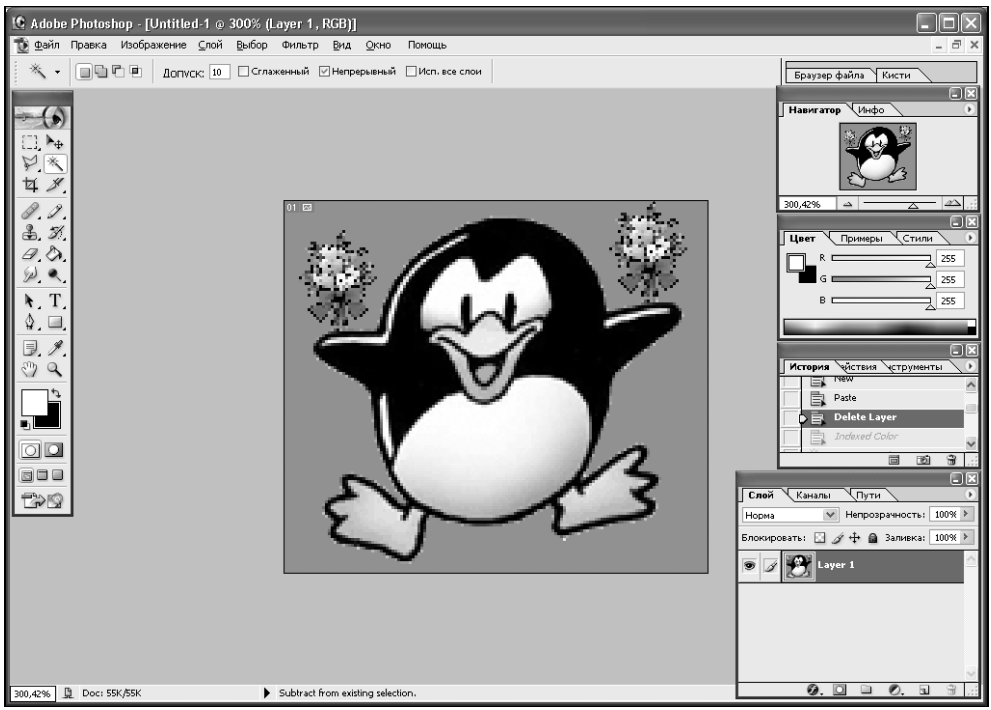


Рис. 4.8. Один кадр персонажа в программе Adobe Photoshop

Подготовьте снимки персонажа таким образом, чтобы он располагался на ярком фоне такого цвета, который не используется самим персонажем, а также близкими ему оттенками и проследите, чтобы границы рисунка были

четкими. Если эти условия не выполняются, то, используя такие инструменты, как **Кисть** и **Заливка**, постарайтесь исправить имеющиеся недоработки. Для цвета фона (он же будет и цветом прозрачности (Transparent Color)) рекомендую использовать цвет с параметрами R = 255, G = 0, B = 255, т. к. этот цвет — ярко-фиолетовый — неформально является стандартным цветом прозрачности. Конечно, если оттенки этого цвета уже использованы в персонаже, то непременно нужно выбрать другой цвет, тоже, желательно, яркий, чтобы он выделялся на общем фоне и его легко можно было найти в палитре цветов. Перед применением инструмента **Заливка** не забудьте включить "грубую заливку", т. е. выберите инструмент **Заливка** и на панели инструментов снимите флажок **Not roughly** (Не грубо)[2], чтобы сохранить неразмытость границ. Если границы изображения персонажа уже были размыты, то их следует откорректировать.

Так как картинка в данном примере открыта в режиме RGB, то первым нашим действием будет индексирование цветов. Это нужно будет проделать с каждым "неправильным" изображением. Для всех кадров должна применяться одна и та же палитра, т. к. при несовпадении индексов используемых цветов все они изменятся на другие и в результате вместо того, что вы видели на экране в Photoshop, в редакторе персонажей будет совершенно иное цветовое изображение. Так что первой картинкой для индексирования должна быть та, в которой больше всего использовано цветов. Прежде всего откройте выбранное изображение. В меню **Choose Image** (Изображение) в подменю **Mode** (Режим) выберите пункт **Indexed Color** (Индексированные цвета). Откроется диалоговое окно индексирования цветов (рис. 4.9).

Чтобы сохранить качество изображения, оставьте тип палитры **Точная** (Exact) и обратите внимание, что количество цветов не превышает 256. Если количество уникальных цветов будет превышать число 256, то вместо палитры **Точная** придется выбирать один из алгоритмов усреднения цветов: **Восприимчивый** (Perceptual), **Выборочный** (Selective) или **Адаптивный** (Adaptive). Заметьте, что в этом случае идет потеря цветовой гаммы, т. к. количество уникальных цветов подгоняется к количеству 256 и некоторые оттенки заменяются на один цвет. Примените данную индексацию, щелкнув на кнопке **OK**.

После индексирования цветов в наиболее полном по цветовой гамме изображении следует сохранить текущую палитру и затем применять ее ко всем остальным изображениям, чтобы не нарушить индексацию цветов. Поэтому выберите в меню **Изображение** (Choose Image) пункт **Палитра** (Palette) и когда появится окно с вашей текущей палитрой цветов, сохраните ее, щелкнув на кнопке **Сохранить** (Save) в окне палитры.

---

[2] Для Adobe Photoshop v7.0 может называться **Сглаживание**.

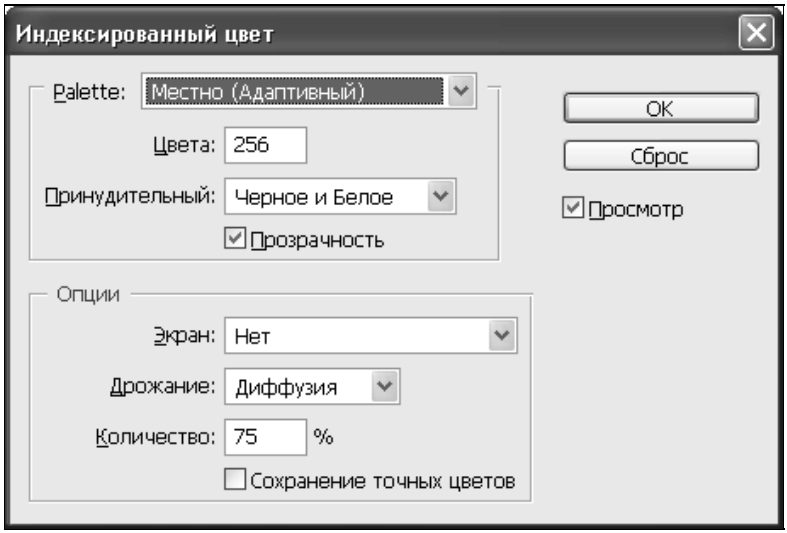


Рис. 4.9. Диалоговое окно индексирования цветов **Индексированный цвет**

Теперь возьмите инструмент **Волшебная палочка** и с его помощью выделите все области фона. Затем удалите фон, нажав клавишу <Delete>. Картинка должна приобрести примерно такой вид, как показано на рис. 4.10. Теперь можно сохранить файл.

Рис. 4.10. Конечный вариант кадра в Adobe Photoshop

Индексируя цвета в остальных кадрах, применяйте к ним сохраненную палитру. Для этого точно так же сначала откройте рисунок, а затем окно индексирования цветов, но выбирайте в списке **Палитра** (Palette) не любой тип палитры, а тип **Пользовательская** (Custom)[3]. В открывшемся окне загрузите

[3] **Выборочная**, **Пользовательская** или **Заказная**, в зависимости от русификации.

сохраненную палитру из файла и примените ее для изображения, щелкнув во всех открытых окнах на кнопке **OK**.

Возможно, читатели знают свои способы и алгоритмы действий, удобные для решения этой задачи. Однако следует помнить, что цель всегда одна — представить изображение в индексированной палитре с числом цветов не более 256, причем у всех изображений (кадров) анимаций персонажа палитра должна совпадать, иначе в итоге вы получите цветовую "кашу" в редакторе ассистентов.

Также обязательно проследите, чтобы все кадры анимаций были одного размера, ведь хотя это и не принципиально, но подгонять положение одного кадра под другой для совпадения положения персонажа не очень удобно.

Создаем анимации

После того как набор картинок будет готов, можно собирать из них наш персонаж.

При открытии редактора мы имеем возможность сразу приступить к созданию необходимых анимаций — практически в любом количестве со всевозможными названиями. Если вы желаете изготовить стандартный персонаж или специально для MS Office, то учитывайте, что к таким персонажам предъявляются определенные требования. Они касаются набора поддерживаемых анимаций, а для помошников Microsoft Office — еще и размера кадра. Все эти параметры можно настроить вручную, но лучше просто заказать шаблон определенного типа персонажа, при этом редактор самостоятельно настроит размер кадра, а также необходимые заготовки под анимации и их связи с определенными состояниями. Для этого нужно открыть в меню **File** (Файл) пункт **New** (Новый) и в подменю выбрать один из пунктов: **Custom Character** (Пользовательский персонаж), **Default Character** (Персонаж по умолчанию) (т. е. персонаж со стандартными мультипликациями) или **Office Assistant Character** (Помощник для Office).

Об используемой палитре цветов в изображениях персонажа мы уже знаем, поэтому перед тем как приступить к работе над мультипликациями, следует указать палитру цветов и цвет прозрачности. Затем выделите пункт **Animations** (Анимации) в дереве элементов в левой части главного окна редактора (см. рис. 4.5), при этом вид правой части окна изменится, и вкладка **Properties** (Свойства) вместо текущих свойств персонажа предоставит элементы для конфигурирования свойств мультипликаций (рис. 4.11).

В группе элементов **Default Frame Settings** (Исходные настройки кадра) находятся элементы установки размера кадра, а также поле со списком **New Frame Duration** (Новый кадр продолжить через), показывающее для всех создаваемых кадров время по умолчанию, спустя которое данный (новый) кадр должен проигрываться после предыдущего.

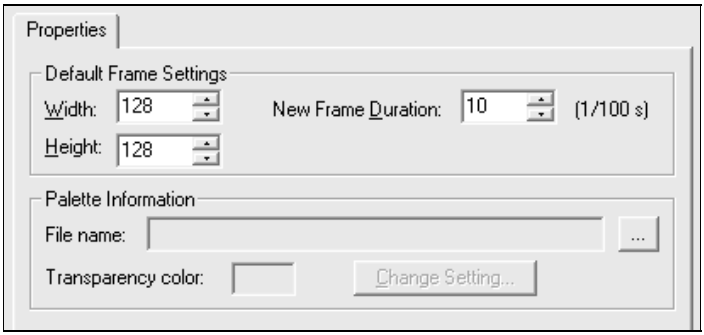


Рис. 4.11. Вид вкладки **Properties** для конфигурации анимаций персонажа

Теперь для нас важно установить нужные значения в группе элементов **Palette Information** (Информация о палитре). Так как используемая палитра цветов должна быть идентична во всех снимках (иначе смысла ни в ней, ни в этой опции не будет), то, если это условие выполняется, в поле **File name** (Имя файла) можно указать имя файла любого изображения персонажа (ведь палитра везде одинакова). Редактор автоматически выберет цвет прозрачности. Правда, делает он это не всегда правильно, поэтому, если в поле **Transparency color** (Цвет прозрачности) указан неверный цвет, щелкните на кнопке **Change Setting** (Поменять установки) и в появившемся окне **Change Transparency Setting** (Поменять установки цвета прозрачности) установите реальный цвет прозрачности (рис. 4.12).

Наконец-то можно приступить непосредственно к созданию необходимых мультипликаций. Для добавления анимации нажмите на кнопку на панели инструментов **New Animation** (Новая анимация) с символическим изображением куска киноленты. В дереве элементов в левой части главного окна редактора появится новый узел, обозначенный значком с тем же кусочком киноленты. При выделении анимации в дереве в правой части окна отобразятся ее свойства. Среди них название анимации (Animation Name), которое можно отредактировать, ее тип (Return Animation), определяющий, является ли она возвратной для какой-либо другой анимации, а также список самих состояний, с которыми она связана. Эти опции мы еще рассмотрим, а пока перейдем к созданию фреймов (Frames).

Все анимации состоят из отдельных кадров — фреймов, добавляются они к анимации щелчком мыши на кнопке **New Animation Frames** (Новые кадры анимации), расположенной справа от кнопки **New Animation**. Новый кадр присоединяется в виде дочернего узла к выделенной анимации. Только что созданный кадр является пустым и не содержит никакого изображения, поэтому в правой части главного окна на вкладке **General** (Главная) в группе элементов **Images** (Изображения) нужно нажать кнопку **Add Image** (Добавить

картинку) и в диалоговом окне открытия файла выбрать необходимую картинку (рис. 4.13). Все просто, однако довольно неудобно для создания кадра каждый раз нажимать на кнопку, а затем загружать картинку — проще пользоваться пунктом **New Frames from Images** (Новые кадры из картинок) контекстного меню анимации в дереве элементов левой части главного окна. При выборе группы картинок в окне открытия файла для каждого изображения в этом случае создается отдельный кадр.

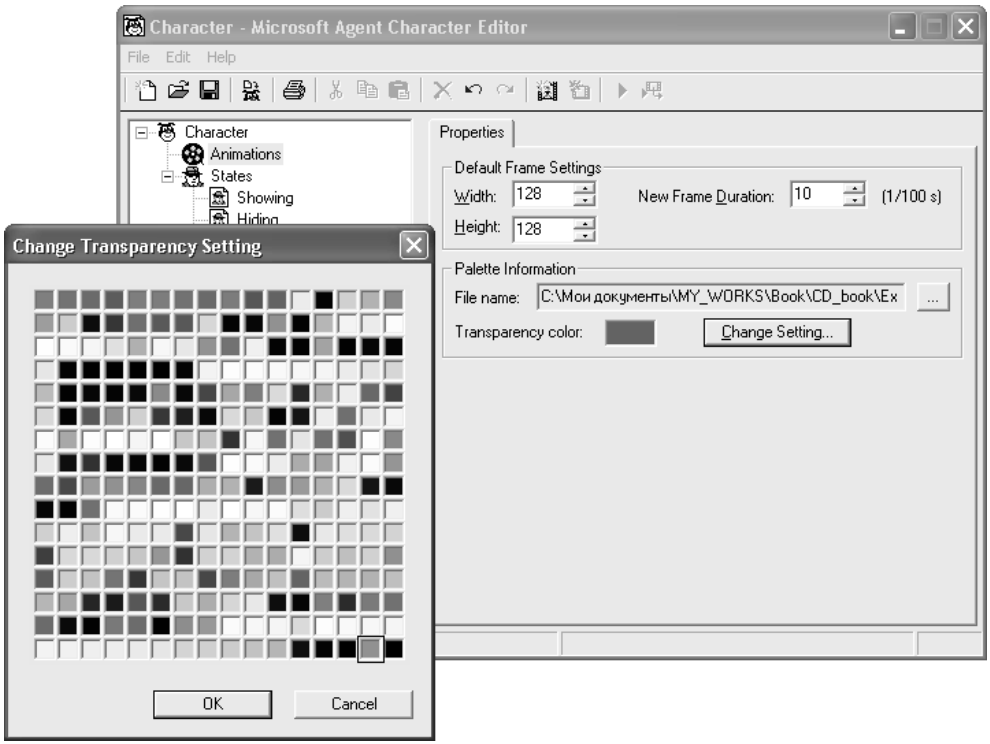


Рис. 4.12. Окно выбора цвета прозрачности для палитры в редакторе персонажей (этот цвет не всегда самый последний)

В группе элементов **Duration** (Через) на вкладке **General** каждого кадра указывается промежуток времени ожидания вывода следующего фрейма в сотых долях секунды. По умолчанию он имеет длину, указанную в свойствах узла **Animations** (Анимации).

Подобным образом разрабатываются все необходимые мультипликации для персонажа. В ходе их создания в любое время доступны для просмотра промежуточные результаты. Щелчком на кнопке **Preview** (Просмотр) на панели инструментов с изображением значка *play*, как на магнитофонах, можно за-

пустить любую последовательность кадров и посмотреть, насколько ваш замысел совпадает с действительностью. Кадры будут отображаться в том же окне, для предпросмотра анимации в отдельном окне через команду **Edit | Open Frame Window** (Редактировать | Открыть окно кадра) можно открыть и дополнительное окно с выводимой анимацией (см. рис. 4.13).

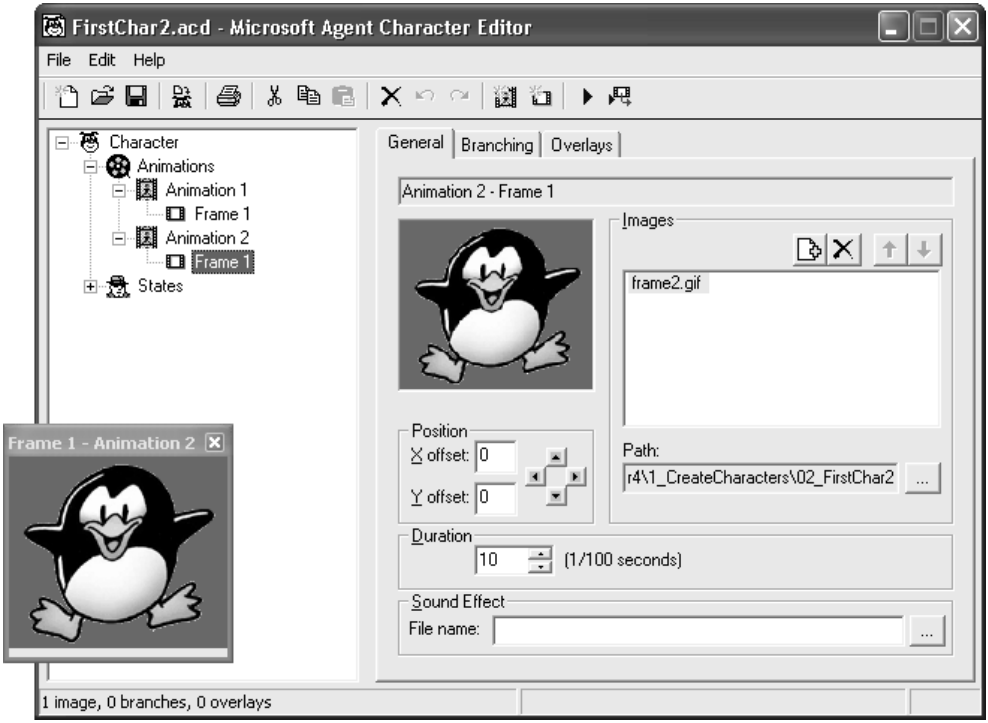


Рис. 4.13. Редактор персонажей с двумя созданными анимациями, включающими каждая по одному кадру

Чтобы закончить создание простейшего персонажа, нужно сохранить проект для будущего редактирования, а потому выполните команду **File | Save as** (Файл | Сохранить как). Персонаж будет сохранен во внутреннем формате ACD, представляющем собой, по сути, описание настроек и ссылок на файлы изображений персонажа.

Здесь одновременно и плюсом, и минусом является то, что редактор сохраняет абсолютные пути на все ресурсы агента. То есть если вы храните исходные картинки будущего агента и сам проект в одном каталоге, то, переместив этот каталог куда-либо, вы можете столкнуться с тем, что редактор при следующем открытии файла не найдет все необходимые изображения и другие файлы. Этого не случится при перемещении единственного файла

проекта, ведь тогда сохраняется полная его работоспособность. Выходом из подобной ситуации может стать только ручная правка ACD-файла. Но не указывать же в программе полные пути ко всем изображениям заново, тем более если персонаж содержит больше сотни кадров? Так что для решения этой задачи откройте исходный ACD-файл проекта персонажа в любом текстовом редакторе, например, в Блокноте, и исправьте все абсолютные пути к ресурсам на относительные. Однако не стоит пользоваться данным приемом для доступа в верхний каталог через ../, такой фокус редактор почему-то не понимает, здесь можно указывать только более глубокие папки относительно текущей, либо просто указывать файлы без путей, т. е. лежащие в каталоге вместе с проектом.

Завершающим штрихом в создании персонажа является его сборка, или компиляция. Для этого выберите пункт меню **File | Build Character** (Файл | Собрать персонаж). Открывшееся окно сборки персонажа похоже на окно сохранения файлов, здесь также нужно указать формат файла — структурный ACS или раздельный (ACF+ACA файлы). По желанию можно снять флажок **Compress** (Компрессия), однако эта опция нам не мешает, и, безусловно, рациональнее пользоваться компрессией файлов персонажа. После выполнения компиляции в указанном месте появится соответствующий файл или файлы персонажа.

Рис. 4.14. Пример работы первого персонажа в проекте 24_GetAnim2 из *гл. 2*

В проекте каталога 1_CreateCharacters\01_FirstChar реализован простейший персонаж с одной анимацией и одним кадром (рис. 4.14). В проекте каталога 02_FirstChar2 (того же подкаталога — 1_CreateCharacters) продемонст-

рировано, как два разных по цветовой гамме, но с одинаково индексированной палитрой цветов изображения используются в персонаже, причем тут сохраняется корректность отображения цветов. Проект персонажа 03_LongAnim (файл проекта находится в каталоге 1_CreateCharacters) демонстрирует довольно продолжительную анимацию, составленную из небольшого числа кадров. В этом примере показано многократное использование одних и те же изображений для повторяющихся кадров. Оригинальные файлы формата BMP, индексированные кадры, а также файл палитры прилагаются.

Увидеть функционирование новоиспеченного персонажа можно в просмотрщике анимаций персонажа из примера каталога 24_GetAnim2 *гл. 2* или с помощью подобного приложения каталога TestChars *гл. 4*.

Привязка анимаций к состояниям персонажа

Мы рассмотрели процесс создания простых агентов с набором анимаций из нескольких кадров. При работе они ведут себя немного не так, как обычные персонажи. Например, они не появляются при отображении методом `Show` и вообще не реагируют на некоторые действия (правда, анимации все-таки проигрываются методом `Play`). Поэтому сейчас мы перейдем к настройкам воспроизведения анимаций, а также коснемся других интересных моментов в создании кадров персонажа.

Наш собственный персонаж не появляется при отображении методом `Show` оттого, что мы не ассоциировали необходимую анимацию отображения персонажа с определенным состоянием. Так что давайте создадим анимацию с любым именем, например, MyShowing, которая будет отображать персонаж. После этого данную анимацию нужно связать с состоянием Showing (Появление), для чего выделите пункт анимации в дереве элементов левой части главного окна редактора и на вкладке свойств анимации в группе **Assign to State** (Связь с состоянием) отметьте флажками те состояния, с которыми должна ассоциироваться эта анимация, т. е. в данном случае установите только флажок **Showing** (Появление) (рис. 4.15). Это будет означать, что при переходе агента в состояние его отображения (при вызове метода `Show`, щелчке на пиктограмме и т. д.) будет воспроизводиться анимация MyShowing. Я специально предложил в качестве примера создать анимацию не с классическим именем Show, чтобы продемонстрировать, что название мультипликации ничего не решает, а все зависит от связей. В демонстрационном персонаже с именем 04_AssignState реализованы два состояния: Showing (Появление) и Hiding (Скрытие). Скомпилируйте персонаж и посмотрите, как он работает в приложениях. При скрытии и отображении ассистента теперь проигрывается назначенная анимация.

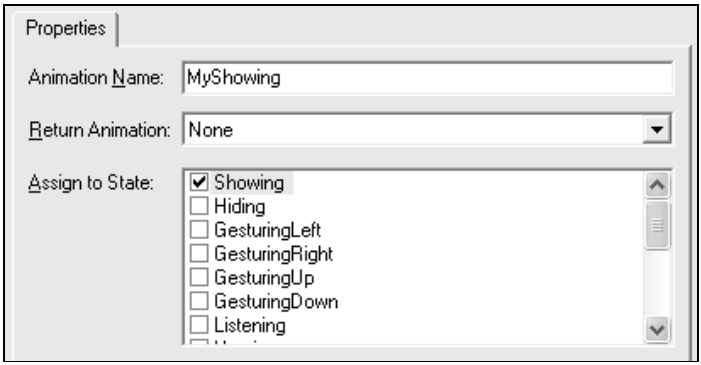


Рис. 4.15. Установление связи анимации MyShowing с состоянием Showing

Из построения интерфейса редактора персонажей можно понять, что одна анимация может быть ассоциирована с несколькими состояниями одновременно — и это действительно так. Кроме того, с одним состоянием может быть связано более одной анимации, и сервер в таком случае будет случайным образом выбирать одну из них. Для того чтобы назначить более двух анимаций одному состоянию, нужно в группе флажков **Assign to State** (Связь с состоянием) для обеих анимаций отметить одно и то же состояние. Можно и наоборот, выделить нужное состояние в дереве, а затем на вкладке **General** (Главная) отметить все анимации, которые следует связать с данным состоянием.

Анимации возвращения (Return Animation)

Одним из недостатков разбиения движения персонажа на анимации является возникновение эффекта беспорядочного перепрыгивания с одной анимации на другую и, соответственно, появления "дерганий" агента. Скорее всего, читатели, работая с качественными персонажами, например, с агентами от Microsoft, заметили, что в каком бы порядке не вызывались анимации, персонаж почти всегда двигается плавно, во всяком случае, без резких движений. Плавно достает листок бумаги (Genie) или книгу (Merlin) во время чтения, также плавно убирает обратно при вызове другой анимации. Один из способов поддержки мягких переходов между анимациями состоит в том, чтобы заканчивать каждое действие в нейтральном положении. Однако это не всегда удобно, ведь некоторые анимации не имеют смысла, если заканчиваются в нейтральном положении. Как раз для таких ситуаций используются *анимации возвращения* или *возвратные анимации* — Return Animation. Они представляют собой обычные анимации, назначенные как возвратные для каких-либо других мультипликаций.

Для использования возвратной анимации создайте необходимую анимацию обычным способом, затем так же создайте целевую анимацию (т. е. ту анимацию, для которой будем создавать возвратную). Теперь остается только связать их. На вкладке свойств целевой анимации в раскрывающемся списке **Return Animation** (Возвратная анимация) выберите название мультипликации, которую вы планировали на роль возвратной (рис. 4.16).

Возвратная анимация будет проигрываться всякий раз, как персонаж должен будет выполнить очередное действие (анимацию) или перейти в другое состояние. Возвратная анимация также выполняется, если целевая мультипликация была запущена сервером вследствие того, что она связана с определенным состоянием.

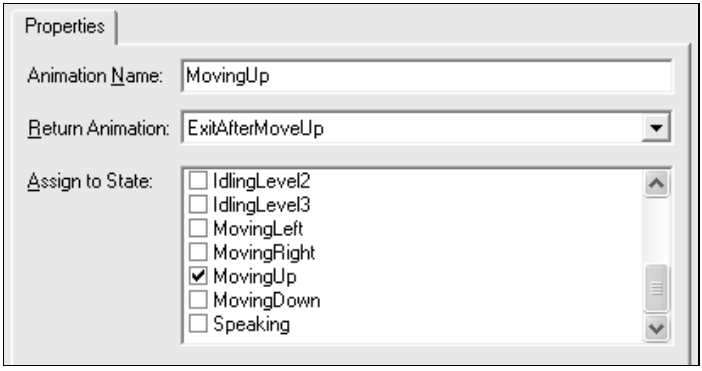


Рис. 4.16. Задание свойств анимации MovingUp, связанной с возвратной анимацией ExitAfterMoveUp

В примере персонажа (файл проекта 05_ExitAnim в каталоге 1_CreateCharacters) целевой является анимация MovingUp, а возвратной — ExitAfterMoveUp. Скомпилируйте персонаж и посмотрите на его работу в приложениях. При выполнении анимации MovingUp персонаж поднимает обе руки и застывает в этом положении. Если после этого запустить любую другую анимацию, например, RestPose или Wave, то прежде чем ее выполнить, ассистент опустит руки, т. е. выполнит анимацию ExitAfterMoveUp. Точно так же агент поступает, если анимация MovingUp выполняется в результате перехода персонажа в состояние MovingUp, т. е. в случае перемещения вверх посредством вызова метода `MoveTo`.

Переходы (Branching)

Мы уже знаем, как заставить сервер выбирать случайным образом анимации определенного состояния для воспроизведения. Достигается это указанием для одного состояния нескольких анимаций. Все это неплохо, однако агент

имеет не так много состояний, а хотелось бы как-нибудь менять ход анимации, добавляя элемент случайности. Реализует данную возможность принцип переходов — Branching.

Каждый кадр анимации, если вы заметили, помимо вкладки свойств, имеет вкладку **Branching** (Переходы), на которой конфигурируются переходы (рис. 4.17). В верхней части вкладки указывается имя данной анимации и номер текущего кадра, это удобно, т. к. помогает не запутаться и не забыть, что за чем следует.

В группе элементов **Normal Branching** (Обычные переходы) указывается (в процентах) вероятность перехода этой анимации на другие кадры, причем возможны три ветвления. Например, можно указать, что в 60% случаев воспроизведения при достижении текущего кадра осуществляется переход на кадр под номером 1, в 15% случаев — на 5-й кадр и в 25% случаев — на 10-й, при условии, что мультипликация содержит не менее 10 кадров (в противном случае редактор не допустит установки данного значения). Естественно, не обязательно указывать все три ветки переходов — в оставшихся необозначенными точной цифрой процентов вероятности случаях анимация будет продолжаться со следующего кадра.

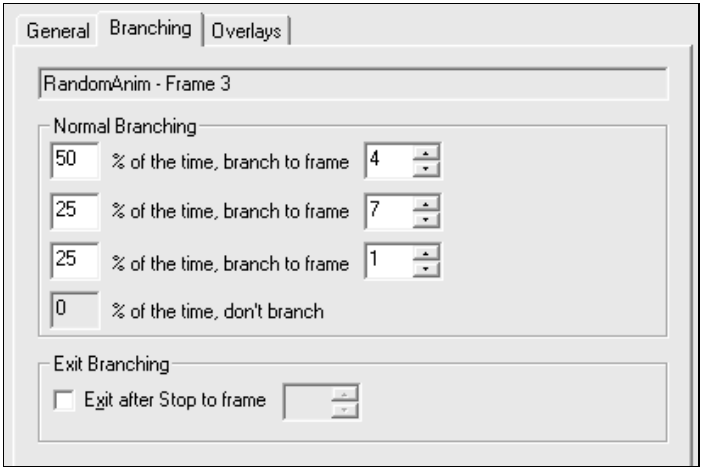


Рис. 4.17. Вкладка **Branching** главного окна редактора персонажей

Посмотрите, как воспроизводится случайная анимация в проекте персонажа 06_RandomFrame (каталог 1_CreateCharacters[4]). Иногда он долго машет рукой, но временами его махание ограничивается лишь одним циклом подня-

[4] Далее указываю только название файла, т. к. все они находятся в одном каталоге — 1_CreateCharacters.

тия и опускания руки. Разумно пользуясь данной возможностью и выстраивая определенным образом ряд кадров, можно добиться поразительных эффектов. Персонаж станет еще более естественным, а поведение его — близким к поведению человека.

На принципе переходов основано создание циклических анимаций, таких как, например, стандартная анимация Processing, длящаяся до тех пор, пока действие не будет остановлено методом `Stop`. Обычно для этого необходимо установить стопроцентную вероятность перехода с последнего кадра на первый. Посмотреть работу и устройство циклической анимации можно в проекте персонажа 07_CircleAnim. Останавливается циклическая анимация методом `Stop` с указанной ссылкой на соответствующий объект `Request`, а также методом `Stop` без параметра (т. е. с переменной `EmptyParam`) и `StopAll` с параметром `Play`.

Возвратную анимацию мы научились создавать в предыдущем подразделе, но есть еще один способ создания анимационного ряда, предназначенного для вывода персонажа в нейтральное положение. Он состоит в использовании группы элементов **Exit Branching** (Заключительные переходы) любого кадра анимации, расположенной на вкладке **Branching** (см. рис. 4.17). Если в этой группе установить флажок **Exit after Stop to frame** (Выходить после остановки на кадр), то можно будет определить номер кадра, на который будет переходить анимация при ее остановке. Под остановкой подразумевается прекращение анимации специальными методами элемента управления Agent Control, а также через сервер при переходе в другие состояния. Данную опцию удобнее всего применять, если анимация достаточно длинная или имеет циклическую структуру, а выходная анимация — непродолжительная. В этом случае создание дополнительной возвратной анимации не всегда оправдано, особенно при большом числе мультипликаций.

Заключительные переходы используются в проекте 07_CircleAnim для возвращения из цикла и в персонаже 05_ExitAnim при завершении анимации Wave.

Многокартинные кадры

Во времена "ручной" анимации, когда все мультфильмы рисовались художниками на основе отдельных кадров, нередко использовался следующий прием. Часть кадра, остававшаяся на какой-то период действия постоянной, рисовалась как отдельный фрагмент, и на нее накладывались лишь небольшие изменяющиеся в этот период куски. Создавая анимированных ассистентов MS Agnet в редакторе персонажей, мы занимаемся фактически тем же самым. Отдельные кадры, выдаваемые на экран с маленькой задержкой по времени, создают впечатление действия. Создав значительную часть кадров какой-либо мультипликации, можно заметить, что во многих случаях некоторые фрагменты персонажа остаются постоянными — например, когда

персонаж незначительно поворачивает голову в стороны, его нижняя часть обычно не изменяется. Учитывая, что каждая картинка добавляет к конечному размеру файла персонажа свои килобайты, нерационально использовать полноразмерные изображения для составления мультипликации. В таком случае кадры разбиваются на отдельные изображения.

Рассмотрим анимацию DoMagic подбрасывания мячика ногой персонажа Человечек. Фрагмент ее изображен на рис. 4.18.

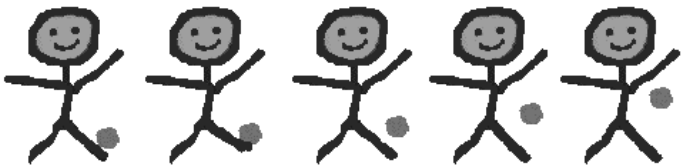


Рис. 4.18. Ряд кадров анимации DoMagic для демонстрационного персонажа Человечек

В подобных анимациях нетрудно заметить повторяющиеся части. В данном случае туловище Человечка без одной ноги остается постоянным, в положениях его левой ноги можно выделить два-три состояния, а мяч меняет лишь свое положение на экране. Поэтому наиболее практичным здесь будет создать один кадр туловища персонажа без левой ноги, ряд движений левой ноги, и изображение мяча (рис. 4.19).

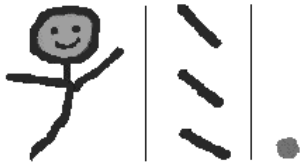


Рис. 4.19. Оптимизированный ряд изображений для создания анимации рис. 4.18. Слева направо: туловище, анимации ноги и мяч. Даже визуально чувствуется, что такой ряд гораздо меньше "раздует" размер конечного файла персонажа

Теперь наша задача — объединить картинки в один кадр. Для этого обычным способом создайте новый кадр в анимации, затем добавьте в него не одно изображение, как мы это делали раньше, а все три: туловище, одно изображение ноги и мяч. Имена файлов появятся в списке группы элементов **Images** (Изображения) на вкладке **General** (Главная) (рис. 4.20). Их следует расположить в нужном порядке по оси координат z (воображаемой оси,

перпендикулярной плоскости экрана). То есть то изображение, которое выше к началу списка, будет располагаться ближе к нам. Эту особенность следует учитывать, если изображения должны перекрывать друг друга.

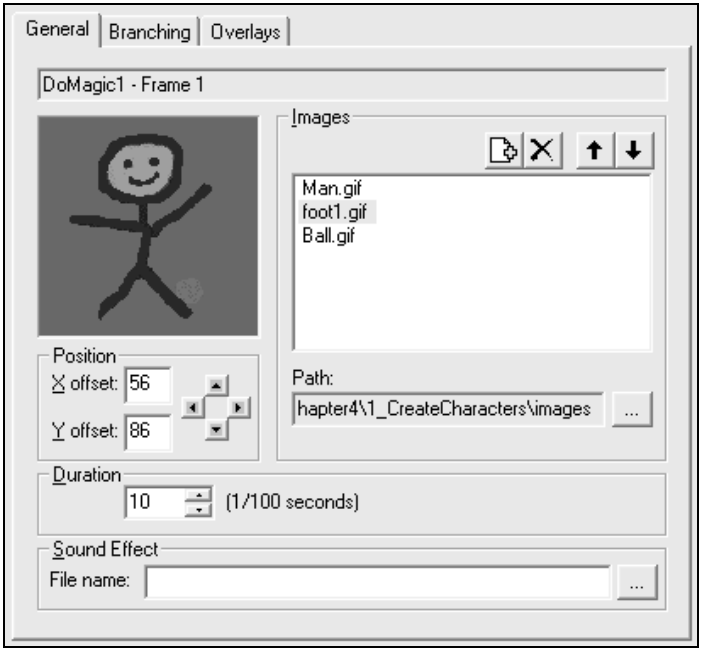


Рис. 4.20. Кадр, состоящий из трех отдельных изображений

Для небольших частей кадра вовсе не требуется создавать изображение, соответствующее размеру кадра. Положение маленькой картинки подгоняется указанием смещения по координате X и Y в группе элементов **Position** (Позиция).

Пример персонажа в файле проекта 08_MultiImageFrame реализован так, как описано выше. Откомпилируйте проект, посмотрите на размер созданного файла и сравните его с размером файла предыдущего примера. Данный агент имеет размер порядка 20 Кбайт, а предыдущий, хотя составлен из меньшего числа кадров анимаций, — около 28 Кбайт. Оптимизация размера файла и удобство создания анимационного ряда налицо. Может быть, выигрыш в 8 Кбайт и невелик, однако не забывайте, что качественные персонажи, использующие многоцветные изображения и продолжительные анимации, имеют размер до 3—5 Мбайт, а оптимизация кадров анимаций может уменьшить его до 2—4 Мбайт.

Звуковые эффекты

В анимацию персонажа возможно вставлять звуковые эффекты в виде волнового файла формата WAV или специального LWV-файла, подготовленного в лингвистическом редакторе.

В кадре, с которого должно стартовать воспроизведение звука, в единственном текстовом поле группы элементов **Sound Effect** (Звуковой эффект) следует всего лишь указать имя звукового файла.

При использовании звуковых эффектов нужно соблюдать некоторую осторожность. Во-первых, нелишним будет как следует оптимизировать звуковой файл, чтобы конечный файл персонажа не "растолстел" до неприемлемого размера, например, уменьшить число каналов до 1, количество бит до 8, а частоту дискретизации до 22,5 кГц.

Другой аспект использования звука, на который следует обратить внимание, — это расчет длины звукового эффекта. Здесь важно, чтобы звук одного файла не наслаивался на другой, иначе нарушится синхронизация анимации со звуком. Кроме того, время воспроизведения звукового файла не должно превышать продолжительность анимации, т. к. сервер не посылает событие окончания анимации до тех пор, пока продолжается воспроизведение звука. Также не следует внедрять спецэффекты в анимации, связанные с состояниями Listening (Слушает) и Hearing (Услышал), т. к. посторонние звуки могут помешать распознаванию речи.

В примере персонажа из файла проекта 09_SoundEffect реализована анимация из предыдущего примера, но уже с использованием звукового эффекта.

Анимации рта

Для того чтобы во всеоружии приняться за подготовку полноценных персонажей, нам остается рассмотреть принципы создания анимации рта. При синтезе речи методом `Speak` "говорящий" персонаж должен открывать рот в соответствии с воспроизводимыми звуками.

В документации MS Agent SDK (раздел *Agent States*) описан набор стандартных анимаций, а также обозначены мультипликации, в которых используется анимация рта, т. к. персонаж может синтезировать речь не в ходе любого действия (ссылку см. в *приложении 1*). Однако даже в обозначенных анимация он не выполняет эти действия синхронно, агент говорит только при воспроизведении последнего кадра, т. е. кадр воспроизводится до тех пор, пока длится речь. Предварительно для последнего кадра нужно указать набор изображений, которые были бы ассоциированы с разным положением губ при произнесении определенных звуков. В табл. 4.1 описаны основные положения рта, которые используются персонажем в ходе синтеза речи.

Таблица 4.1. *Описание положений рта персонажа в ходе синтеза речи*

Положение рта	Пример	Описание
Закрытый		Закрытый рот. Также используется при произнесении звуков "m", "f", "b" и т. д.
Широко открытый 1		Слегка открытый, в полную ширину. Для фонем "g", "i", "ear" и т. д.
Широко открытый 2		Частично открытый по полной ширине. При звуках "n", "d", "t" и др.
Широко открытый 3		Открытый рот в полную ширину. Звуки такие, как "u", "ea", "ur" и т. д.
Широко открытый 4		Полностью открытый рот в полную ширину. Для таких фонем, как "a", "ow" и др.
Средне-открытый		Открытый рот в половину ширины. Звуки "oy", "o" как в слове "hot"
Узкий		Узко открытый рот. Для звуков "o", как в слове "hoop", "w" и т. д.

В таблице представлены англоязычные примеры фонем (или звуков), автор намеренно это сделал, чтобы не импровизировать по поводу русскоязычных аналогов. Однако анимации рта так же хорошо работают и при воспроизведении русской речи, о чем свидетельствует демонстрационный пример персонажа в файле проекта 10_Mouths.

Для того чтобы указать нужные изображения рта персонажа, накладываемые на определенную анимацию, следует выбрать любой кадр анимации (как мы знаем, имеет значение только последний), перейти на вкладку **Overlays** (Перекрытие) главного окна редактора персонажей и указать с помощью группы элементов **Type** (Тип) для каждого положения губ соответствующую картинку (рис. 4.21).

Для каждого изображения рта, так же как и в случае указания нескольких частей кадра персонажа, в группе элементов **Position** (Позиция) настраивается смещение изображения картинки по координатам X и Y.

По умолчанию изображения рта выводятся поверх всего базового кадра, однако в определенных случаях необходимо изменить этот порядок. Выводить кадр поверх изображения рта может понадобится в случае, когда персонаж говорит, например, повернувшись к пользователю боком. В этих и других ситуациях следует для определенного положения губ установить флажок **Replace base frame top image** (Заменить базовый кадр верхним изображением) или нажать на кнопку **Set all** (Установить для всех), чтобы установить указанное положение опции для всех типов губ.

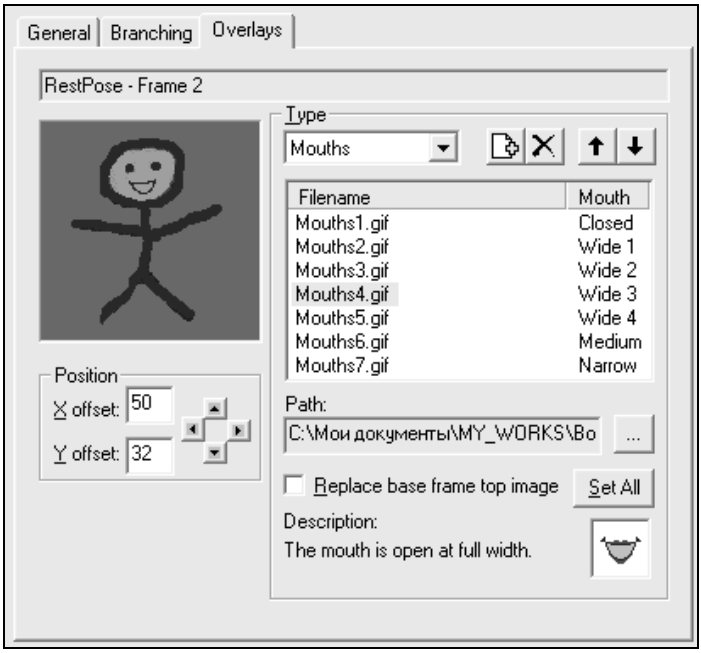


Рис. 4.21. Вкладка **Overlays**, на которой указываются изображения состояния губ персонажа

На рис. 4.22 показан наш самодельный персонаж из примера файла проекта 10_Mouths во время чтения им текста. Посмотрите на движения губ, они вполне адекватны читаемым словам, даже с учетом того, что анимационный ряд далек от совершенства.

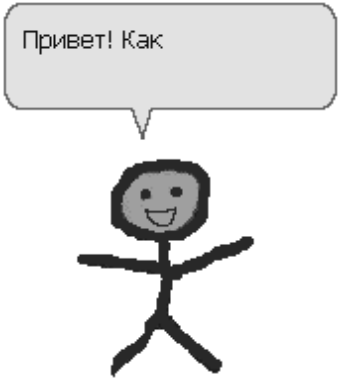


Рис. 4.22. Персонаж с поддержкой анимации рта

Вот мы и познакомились с некоторыми принципами и методами создания анимированного персонажа в программе Microsoft Agent Character Editor. Думаю, читателям было интересно получить информацию, помогающую при создании собственного агента, и они обязательно попробуют произвести на свет какое-нибудь свое анимированное существо.

4.3. Лингвистический редактор Microsoft Linguistic Information Sound Editing Tool

Не раз мы уже применяли волновые (звуковые) файлы для воспроизведения речи или других звуков взамен ее синтеза. И все же при воспроизведении звуковых файлов можно заметить небольшие отклонения в синхронизации губ рта с выводимыми звуками. Для корректировки данной синхронизации корпорация Microsoft разработала специальный лингвистический редактор, в котором можно создавать файлы (LWV), содержащие как волновые данные звука, так и лингвистические данные о фонемах и границах слов. Только подумайте, и как бы мы без этого обошлись? Что ж, давайте посмотрим, что он собой представляет и чего можно добиться с его помощью.

Начало работы

Прежде чем приступить к работе с лингвистическим редактором, его следует установить. Адрес для загрузки редактора вы сможете найти в *приложении 1.* Версия размером в 6,5 Мбайт уже содержит в себе, кроме самого редактора, пакет функций Speech API и движок распознавания речи Microsoft Speech Recognition Engine, поэтому при установке предлагаются для прочтения две пользовательские лицензии (Speech API при установке лицензию не показывает). Лингвистический редактор, так же как и редактор персонажей, по умолчанию желает устанавливаться в папку X:\Program Files\Microsoft Agent\Tools, куда копирует всего один файл Liset.exe.

При запуске редактора появляется главное окно программы, показанное на рис. 4.23.

С помощью лингвистического редактора можно как открывать уже созданные волновые файлы формата Windows sound file (WAV) различного качества, так и записывать звук с микрофона. В отличие от редактора персонажей, программа Linguistic Information Sound Editing Tool открывает уже созданные лингвистические LWV-файлы.

В верхней части главного окна располагается поле редактирования, в которое вводится текст, соответствующий словам в волновом файле. Справа от этого поля находится символическое изображение губ, демонстририрующее,

как будет открываться рот персонажа при воспроизведении данного файла. В нижней части окна на сером фоне отображается сам волновой файл.

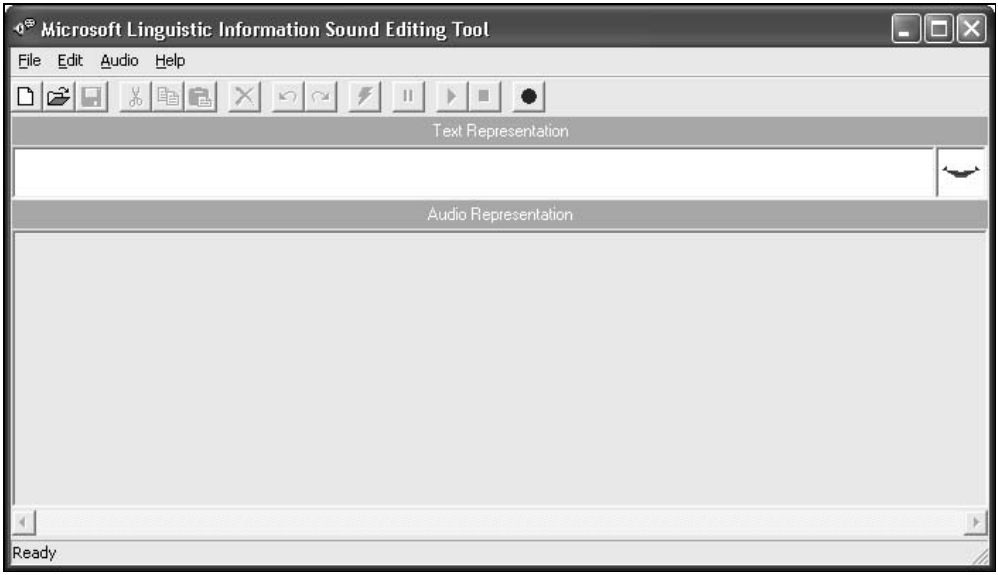


Рис. 4.23. Главное окно лингвистического редактора

Создание лингвистических LWV-файлов

Давайте познакомимся с процессом подготовки LWV-файлов непосредственно на примере.

Откройте в лингвистическом редакторе волновой файл из каталога 2_CreateLWV\file.wav. Замечу, что это файл с английскими словами, т. к. редактор распознает английскую речь, вернее выделяет в волновом файле ее фонемы. Эти действия производятся при помощи модуля распознавания речи, установленного в системе. Так как самым близким из доступных движков является англоязычный, то и распознавать в звуках можно только английские слова. Надеюсь, вы уже открыли файл, и в главном окне лингвистического редактора появился характерный синусоидальный график. Нажав на панели инструментов третью справа кнопку **Play** (Проигрывание) можно прослушать запись слов "This is a message", что означает: "Это сообщение".

Так как лингвистический редактор использует речевые модули для распознавания речи, то следует выбрать один из них. В том случае, когда в системе имеется всего один модуль, эту операцию можно и не выполнять. Одна-

ко если вы все-таки где-нибудь раздобудете несколько движков распознавания речи, вам полезно знать, каким образом выбирается нужный модуль. В меню **Edit** (Редактировать) отметьте пункт **Speech Engine** (Речевой модуль), после чего откроется окно со списком доступных движков, в котором останется только выбрать нужный.

Для того чтобы редактор с помощью движка расставил по местам соответствующие фонемы, необходимо ему помочь в плане ввода произносимого текста. Без этого никаким способом не удастся заставить редактор сгенерировать лингвистическую информацию. Поэтому в верхнем поле **Text representation** (Представление текста) главного окна редактора нужно ввести текст *This is a message*, как показано на рис. 4.24.

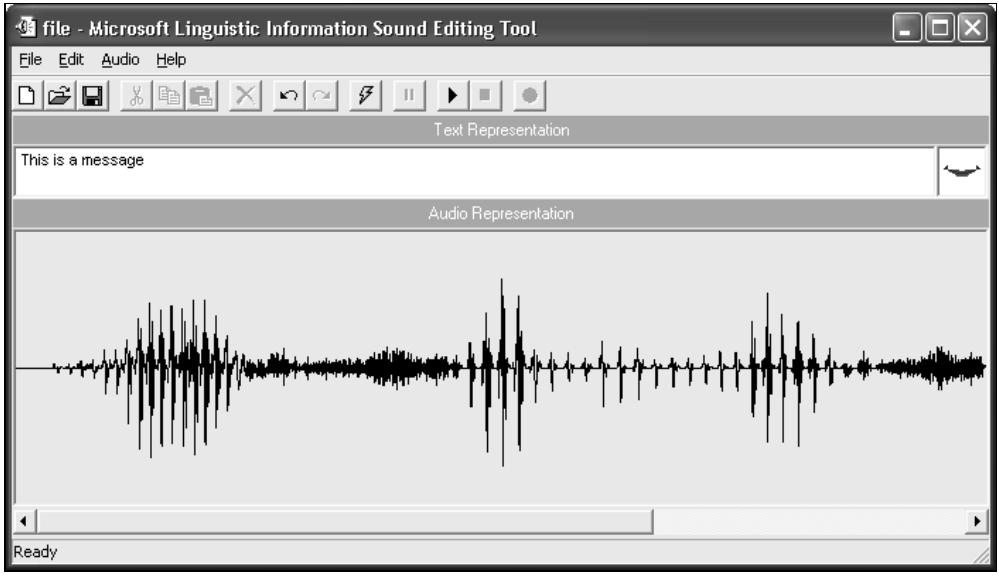


Рис. 4.24. Главное окно лингвистического редактора с открытым волновым файлом

Теперь нажмите на панели инструментов кнопку с изображением искры **Generate Linguistic Information** (Генерировать лингвистическую информацию) и, наблюдая за ходом продвижения индикаторной полосы в новом окне, подождите результат. Вскоре в окне редактора над синусоидальным графиком появятся цветные полосы, обозначающие границы слова, а под графиком — полосы с подписанными фонемами (рис. 4.25).

С английской речью никаких проблем не должно быть, разумеется, если вы написали правильный текст и не поставили лишних знаков препинания или

еще чего-нибудь. Осталось сохранить наши результаты в файл LWV, а затем использовать в любой программе вместе с MS Agent.

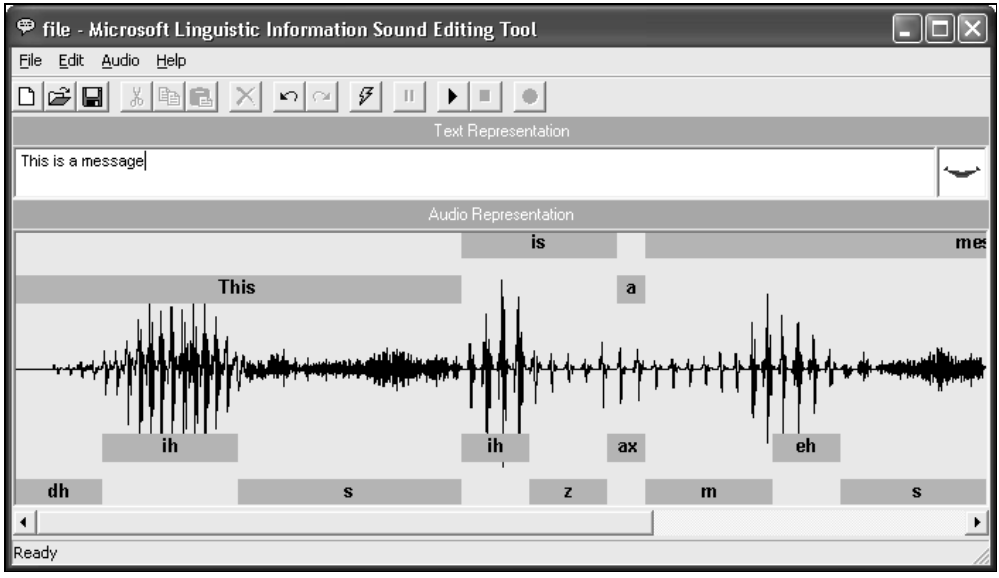


Рис. 4.25. Редактор сгенерировал лингвистическую информацию, над синусоидальным графиком появились обозначения границ слов, под графиком — фонемы

Чтобы проиграть лингвистический файл с помощью персонажа, нужно воспользоваться методом `Speak` персонажа. Например:

```
var
  Chars: IAgentCtlCharacterEx;
begin
  ...
  Chars.Speak('', 'c:\01_Simple.lwv ');
end;
```

Если при вызове метода `Speak` указать только имя лингвистического файла, а параметр текста оставить пустым, то в воздушном шарике Balloon будет показан текст, указанный в лингвистическом редакторе при распознавании фонем. Вместе с тем, набрав собственный текст, можно определить иные слова для отображения в воздушном шарике.

Создание и редактирование фонем и слов

В связи с тем, что в случае английской речи все обстоит просто и чаще всего редактировать ничего не надо — ведь губы персонажей и без лингвистического редактора открываются синхронно с произнесением слов — перейдем к русскоязычным примерам.

Откроем в редакторе Microsoft Linguistic Information Sound Editing Tool волновой файл 2_CreateLWV\msg.wav, где присутствуют слова: "Обратите внимание на сообщение", и т. к. редактор требует ввести оригинальный текст, напишем что-нибудь вроде:

Obratite vnimanie na soobschenie

Теперь, как и в предыдущем примере, заставим программу автоматически определить слова и фонемы. В принципе, вовсе необязательно заставлять делать это редактор, можно и самим создать все слова и фонемы — и мы скоро узнаем, как именно. Однако такой способ потребует намного больше времени, чем может показаться на первый взгляд, и проще будет отредактировать сгенерированные редактором фонемы.

Итак, редактор уже сделал свое дело, и результаты вы видите на экране. Если введенная вами текстовая строка была идентична той, что приведена выше, то разбиение на фонемы будет таким, как показано на рис. 4.26.

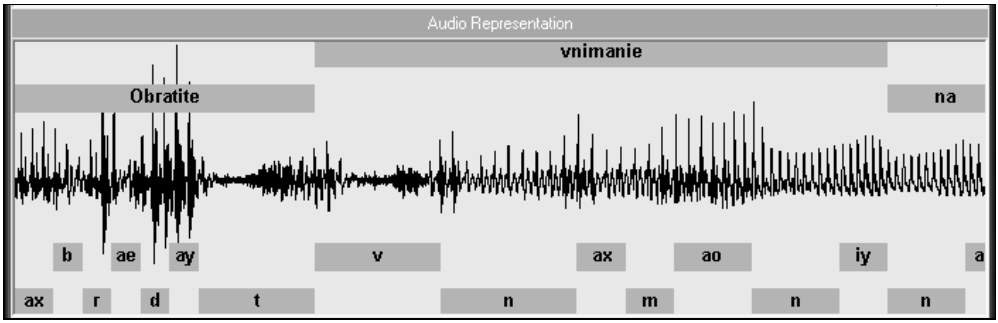


Рис. 4.26. Распознанные слова и фонемы фразы "Obratite vnimanie na soobschenie"

Сразу налицо недочеты: например, в слове "obratite " не определена фонема "ye" (э), а если прослушать первое слово, то окажется, что его граница указана намного раньше, чем наступил реальный конец слова. Так что придется нам отредактировать лингвистическую информацию.

Первым делом следует правильно указать границы слов. Для этого подведите курсор мыши к границам полосы (ленты) слова, чтобы курсор превратился в двунаправленную стрелочку — после этого можно смещать границу в обоих направлениях. Аналогичным образом изменяются и границы фонем.

Если какое-то слово не определено автоматически редактором, его нужно вставить вручную. Этим приемом пользуются также при ручном создании лингвистических данных. Он состоит в следующем. Выделите в главном окне программы с помощью мыши часть звуковых данных, которые звучат как одно слово, вызовите правой кнопкой мыши на выделенном фрагменте контекстное меню, выберите в нем пункт **Insert word** (Вставить слово), в открывшемся диалоговом окне введите нужное слово. Если вы создавали слова на пустом месте, то введенное слово вставится в имеющуюся последовательность. В том случае, когда одно слово (например, "выноски") разбивается на части ("вы носки"), новое слово вставляется между уже существующими, сдвигая их границы. Во втором случае часто приходится изменять одно из двух слов или оба сразу, для чего нужно выделить ленту слова и в контекстном меню выбрать пункт **Replace word** (Заменить слово). Через то же самое контекстное меню можно и удалить ненужное слово.

Точно так же организуется работа с фонемами. Для вставки новой фонемы первым делом в поле редактирования лингвистической информации выделяется часть звукового файла, составляющая, по вашему мнению, отдельную фонему, затем в контекстном меню выбирается пункт **Insert phoneme** (Вставить фонему). С помощью того же меню определенной фонемы подается команда для ее удаления или замены.

Все фонемы при вставке выбираются в диалоговом окне **Insert phoneme** из определенного списка, как показано на рис. 4.27.

В данном окне можно выбрать любую из стандартных фонем, при этом указывается ее аналог в системе IPA (International Phonetic Alphabet, Международный фонетический алфавит), а также обозначить нестандартную фонему (последнее возможно только в системе IPA). По умолчанию все фонемы в окне редактора отображаются в удобном виде, т. е. в виде ее имени, однако профессионалы могут переключить режим отображения на обозначения в систем IPA по команде **Edit | Phoneme Label Display | IPA** (Редактировать | Отображение фонем | IPA).

Проводить все операции со словами и фонемами можно как через контекстное меню, так и через меню **Edit** (последнее, по моему мнению, менее удобно).

В примере каталога 02_msg_en.lwv лингвистический файл уже отредактирован и готов к использованию. Посмотрев его воспроизведение через программу из проекта каталога TestChars, можно заметить, что в выноске Balloon отображаются русские слова, но написанные латиницей (как мы ввели их в лингвистическом редакторе), если мы не указали первый параметр при запуске метода `Speak`. Отображения кириллицей можно добиться прямым указанием необходимого текста, но это по понятным причинам неудобно. Решается данная проблема просто: после окончательной правки лингвистической информации в редакторе нужно заменить каждое слово на русский аналог.

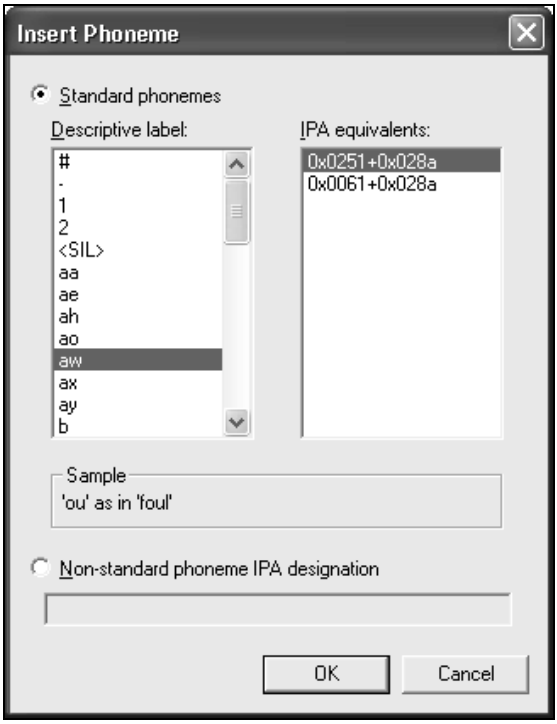


Рис. 4.27. Диалоговое окно выбора фонемы в лингвистическом редакторе

Замечание

В Windows 98 даже в этом случае могут возникнуть трудности — редактор может не позволить вам переключить раскладку клавиатуры по горячей клавише. В этом случае переключитесь через значок раскладки в системном лотке. Русские слова в полях редактирования и в лентах слова иногда превращаются в абракадабру (рис. 4.28), но вы можете убедиться, что в конечном итоге текст в выноске отображается правильно.

В поле ввода **Text Representation** (Представление текста), переключив раскладку мышью, можно попытаться ввести нужный текст непонятными символами, однако в тарабарском виде движок не распознает фонем, а главное для нас — правильное написание слов в воздушном шарике персонажа, и это зависит только он надписей, указанных в словах.

Доработанный файл 03_msg_ru.lwv также присутствует на компакт-диске.

Читатели, вероятно, заметили, что словесная речь практически всегда приемлемо синхронизируется с анимацией губ персонажей. Это действительно так, поэтому для подготовки обычных текстовых фраз лингвистический ре-

дактор и не применяют. Наиболее интересных результатов можно добиться, работая с ним при использовании лингвистических данных для создания фрагментов песен (чтобы заставить ассистента запеть).

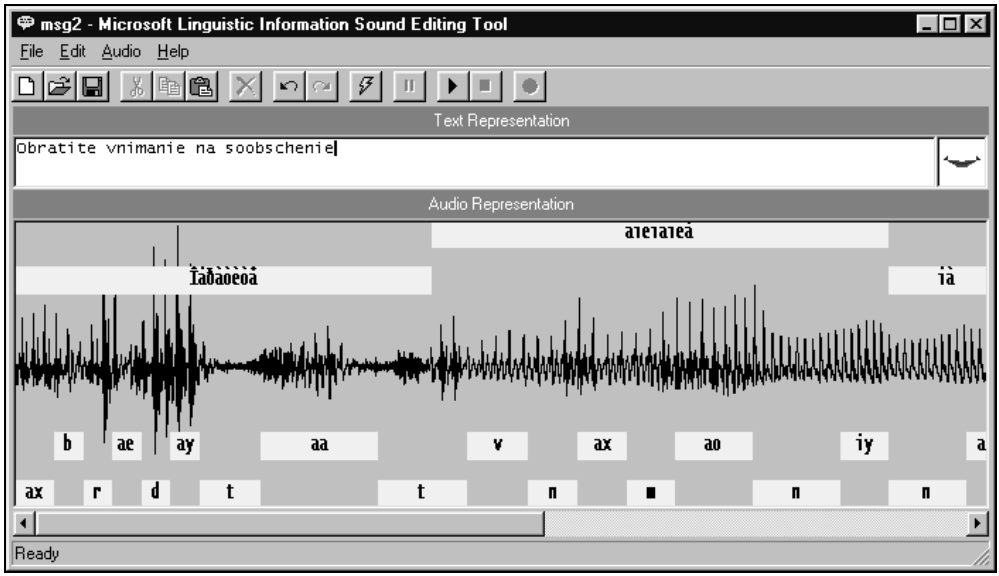


Рис. 4.28. В главном окне лингвистического редактора в Windows 98 подписи и слова отображаются другим набором символов, однако это не мешает правильному отображению слов в Balloon при воспроизведении в рабочих программах

Воспроизведение звуковых файлов формата WAV, включающих, кроме слов, музыкальное сопровождение, приводит алгоритмы синхронизации губ со звуками в замешательство, вследствие чего агент начинает беспорядочно отрывать рот во всю ширь (см. пример из проекта в каталоге 38_Speak *гл. 2*). Поэтому для эффектного пения можно предварительно создать LWV-файл, а затем его воспроизводить. Должен сказать, что даже над небольшим фрагментом придется изрядно потрудиться, т. к. посторонний шум, музыка и прочий лингвистический мусор не дают возможности редактору распознать хоть слово в файле, особенно с русской речью. Все же посмотрите пример из файла 04_Snegur_ru.lwv — насколько теперь движение губ соответствует действительности (рис. 4.29).

Другие тонкости вы сможете узнать, начав самостоятельно работать с лингвистическим редактором, а сейчас мы переходим к еще одной интересной "примочке" к MS Agent.

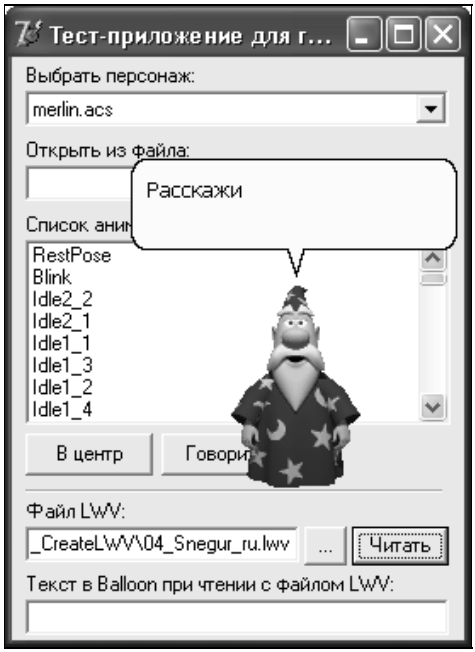


Рис. 4.29. Положение губ персонажа на слоге "Ра" с использованием лингвистических данных

4.4. Компонент BalloonDialog от фирмы SommyTech

Пожалуй, те из разработчиков, кто дочитал книгу до этого места и достаточно активно потренировался в использовании персонажа MS Agent, уже заметили, что стандартная выноска (Balloon) обладает крайне скромной функциональностью, если не сказать больше. Все мы видели выноску в приложениях Microsoft Office, тот воздушный шарик не в пример удобнее и имеет более эстетичный вид в комбинации с MS Agent, чем стандартные диалоги. Поэтому неудивительно, что другие разработчики взялись реализовывать данную идею. В Интернете можно найти довольно много различных реализаций подобных компонентов. Разрабатывают их как иностранные фирмы, так и отечественные, а также независимые разработчики. К сожалению, почти все разработки платные и требуют "энных" сумм рублей или так называемых "улыбающихся енотов" (у. е.). Компонент, о котором сейчас пойдет речь, называется "BalloonDialog от фирмы SommyTech" (**http://www.sommytech.com.ar**) и так же не бесплатен. Однако автор не случайно выбрал именно эту реализацию расширенной выноски — она наиболее функциональна. Почти все не-

зарегистрированные компоненты подобного типа отличаются от лицензионных тем, что при загрузке приложения показывают окно с соответствующим сообщением. Учитывая, что нашим разработчикам все равно сколько НЕ платить денег, 100 у. е. или 100 р. — т. к. нечасто кто-то из них собирается что-то покупать, — я остановил выбор на вышеупомянутом компоненте.

Отмечу еще один факт: почти все типы выносок от разных авторов либо вообще отказываются работать в среде Delphi, либо при запуске приложения, использующего их, "вылетают" с фатальной ошибкой. Практически все они сделаны на Visual Basic и преимущественно для использования в среде Visual Basic. Скорее всего, авторы даже не проверяли свои творения в других, не майкрософтовских средах. Если принять во внимание, что BalloonDialog имеет наибольшее число функций, следует ориентироваться все-таки на него.

BalloonDialog — это компонент ActiveX, который служит для расширения функциональности стандартной выноски персонажа MS Agent Balloon, а также для упрощения взаимодействия персонажа с пользователем. Чтобы долго не интриговать читателей описанием возможностей этого "шаровоздушного" средства, предлагаю обратить внимание на рис. 4.30, из которого станет понятным содержание большинства его функций.

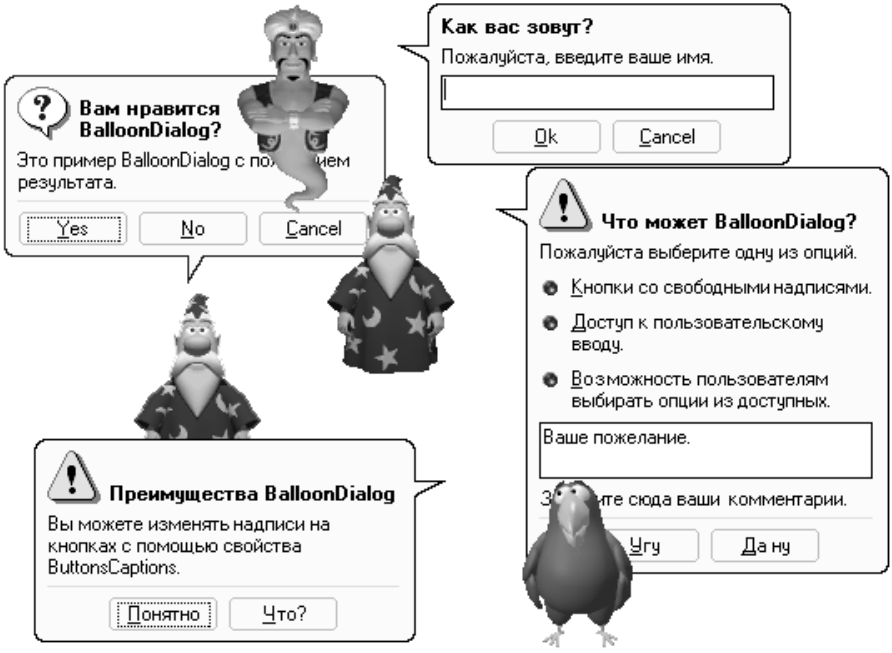


Рис. 4.30. Демонстрация возможностей BalloonDialog

Теперь можно кратко описать эти функции. Выноска заменяет стандартные информационные окна вывода сообщений, диалоги типа `MessageDlg` и `InputBox`; кроме того, надписи на кнопках легко заменяемы. Элемент Balloon может принимать формы, подобные выноскам в приложениях MS Office при вызове справки и отображении интерактивных советов. Еще один тип выноски, который не отображен на рис. 4.30, похож на элемент TipWindow — подсказку. Похожий воздушный шарик появляется в том же MS Office, когда помощник показывает быстрый совет по поводу наших действий.

Ну, а теперь обо всем по порядку.

Начало работы

Несколько слов о самом компоненте. Ссылку на полный дистрибутив (размером 2,6 Мбайт) вы найдете в *приложении 1.* Разработан компонент, судя по всему, итальянской фирмой, т. к. критические ошибки выдаются на итальянском языке. В дистрибутиве, кроме самого компонента, присутствует справочный файл, примеры на Visual Basic, Visual C++ и VBScript.

Инсталлятор включает еще специальную программу регистрации элемента управления. Дистрибутив содержит как Trial-версию, так и полную версию BalloonDialog, правда под "триальностью" подразумевается отображение при старте приложения окна с информацией о разработчиках, компоненте и прочем, в остальном же компонент полностью работоспособен.

Обе версии находятся в дистрибутиве в файлах BlnDialog.ocx и BlnDialogT.ocx. Любопытно, что рабочая версия BalloonDialog никак не спрятана и ее также можно зарегистрировать в утилите regsvr32.exe, однако при последующем создании объекта в приложении выдается сообщение о том, что компонент не лицензирован для использования. Для применения не-Trial-версии нужно получить серийный номер и зарегистрировать ActiveX через программу регистрации.

В Интернете не составит большого труда найти "волшебное слово", которое необходимо сказать регистратору, но, прежде всего, этим мы нарушаем закон об авторских правах, да и приспособившись использовать зарегистрированную версию (я имею в виду: прописав в приложении необходимый CLSID создания компонента или информативное имя), мы привязываемся к зарегистрированной условно-бесплатной версии. Ведь вы же не будете при распространении продукта писать о том, что для отображения всех красивостей нужно купить следующий компонент? Вот только лицензия, не запрещая распространять исходный OCX-компонент, одновременно с этим не позволяет прописывать в системный реестр дополнительные ключи. Поэтому автор нашел несколько экзотический способ, не нарушающий этих пунктов, однако позволяющий пользователю не отвлекаться на дополнительные окна

компонента. Правда, все-таки не нужно забывать благодарить сторонних разработчиков за их компоненты хотя бы в вашей справке.

С BalloonDialog связано еще не менее интересное обстоятельство. Версия 6.5 "криво" устанавливается в Delphi — ни в одну версию среды она не внедряется так, чтобы потом оба элемента — MS Agent и BalloonDialog — уживались вместе. Почему-то один из них исчезает при установке другого, и после этого исчезнувшим элементом можно воспользоваться только через позднее или раннее связывание, но никак не визуальным внедрением. Более того, при удачном внедрении в пакет (иногда это все-таки бывает) компонент отказывается работать: при вызове любой функции приложение "падает" с банальной ошибкой EAccesssViolation. Происходит это только в средах Delphi, хотя в Visual Basic и Visual C++ прекрасно работает, как и в Web. Поэтому мной найдено такое решение: использовать компонент только через позднее связывание. При раннем связывании приложение, использующее BalloonDialog, запускается, но ни одна его функция не работает, хотя никакие ошибки и не выскакивают. Скажу более: не у меня одного "вредничает" элемент управления. До момента написания книги я не нашел в Интернете работающего кода на Delphi, использующего его. Однако если кто-то все же добьется стабильной работы BalloonDialog в среде Delphi с помощью внедрения ActiveX, то прошу написать мне об этом (**subritto@ngs.ru**).

Версии 7.0 и 7.1 компонента BalloonDialog разработаны для платформы .NET Framework, поэтому знакомиться с ними пока имеет смысл только ради любопытства.

Внедряется BalloonDialog как обычный ActiveX, т. е. предлагаемый OCX-файл следует зарегистрировать утилитой regsvr32.exe, дав следующую команду:

```
\>regsvr32.exe BlnDialog.ocx
```

или в случае с Trial-весией:

```
\>regsvr32.exe BlnDialogT.ocx
```

Следует помнить, если OCX-файл находится не в системной папке, то нужно указывать полное имя файла, например:

```
\>regsvr32.exe "C:\OCXs\BalloonDialog\BlnDialogT.ocx"
```

Остальной порядок программирования мы сейчас рассмотрим.

Минимальное приложение

Как мы уже договорились, работать будем в позднем связывании, поэтому никаких дополнительных модулей не потребуется. Единственное, что нам нужно знать, — это информативное имя компонента. Образец кода приве-

ден в листинге 4.1, пример находится в папке 01_BlnAppMin каталога 3_BalloonDialog[5].

Листинг 4.1. Минимальное приложение MS Agent с BalloonDialog в позднем связывании

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dia-
logs, StdCtrls, ComObj;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  end;

var
  Form1: TForm1;
  Agent, Chars: OleVariant;
  Bln: OleVariant;

implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  Agent:=CreateOleObject('Agent.Control.2');
  Agent.Connected:=1;
  Agent.Characters.Load('Agent','genie.acs');
  Chars:=Agent.Characters.Character('Agent');
  Chars.Show(0);
```

[5] Все проекты этого раздела находятся в подкаталогах директории 3_BalloonDialog, поэтому далее будут указываться лишь папки проектов.

```
  Bln:=CreateOleObject('BlnDialogT.Balloon');
  Bln.ResetProperties;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
 strText: String;
 strTitle: String;
begin
 strText := 'Привет! Это пример минимального приложения BalloonDialog
вместе с MS Agent в позднем связывании.';
 strTitle := 'Привет из BalloonDialog';

 Bln.MsgBalloon(strText, 64, strTitle, Chars);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
 Agent.Characters.UnLoad('Agent');
 Agent.Connected:=0;
end;

end.
```

Как мы видим, подключение MS Agent производится обычным способом, а подключение BalloonDialog — как любого OLE Automation сервера в среде Delphi. Строкой:

```
Bln:=CreateOleObject('BlnDialogT.Balloon');
```

мы получаем указатель на Trial-версию BalloonDialog, если же нужно использовать зарегистрированную версию, то следует указывать другое информативное имя:

```
Bln:=CreateOleObject('BlnDialog.Balloon');
```

В принципе, конечное приложение можно приспособить и для использования обеих версий — например, сначала производить попытку создания нормальной версии компонента, а если ссылка окажется нулевой, то попытаться создать экземпляр Trial-версии.

После создания объекта `Balloon` нужно сбросить его свойства на значения по умолчанию методом `ResetProperties`, иначе даже в позднем связывании ни одна функция работать не будет. После этого можно вызывать любой

метод BalloonDialog. В данном примере (см. листинг 4.1) при щелчке на кнопке **Button1** отображается выноска с одной кнопкой **Ok** (рис. 4.31).

Замечание

Методом `ResetProperties` мы будем довольно часто пользоваться, т. к. после изменения некоторых свойств объекта `Balloon` возвращать их в начальное состояние довольно трудоемко. В реальных приложениях подобное может встретиться редко, т. к. во время запуска настраиваются все важные параметры, такие как вид кнопок, шрифты, цвета и т. д. В то же время в проектах примеров частенько придется приводить значения свойств к начальному состоянию, чтобы изменения в них не отражались на работе всего цикла запуска приложения.

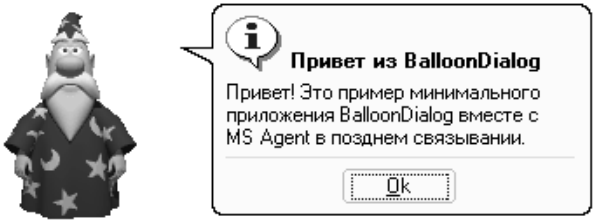


Рис. 4.31. Рабочее окно простейшего приложения с BalloonDialog

Данный воздушный шарик вызывается функцией `MsgBalloon`:

```
function  MsgBalloon(const Prompt: WideString; Buttons: BalloonStyle;
const Title: WideString; const oAgent: IAgentCtlCharacterEx; const
oAgentSvr: IAgentCharacterEx): MsgBalloonResult;
```

Первый параметр функции — это заголовок окна, третий — непосредственно текст, второй параметр объединяет типы кнопок на выноске и тип значка. В данном примере для третьего параметра использовалось значение 64, что соответствует стилю окна со значком информации и одной кнопкой **Ok**. Чтобы не запоминать все константы и не "бегать" по справочной системе Visual Basic для просмотра их значений, можно сразу вставить список необходимых констант (такой же эффект достигается подключением модуля с библиотекой типов от BalloonDialog). Однако не всегда удобно подключать каждый раз к очередному проекту большой модуль, из которого нужны лишь константы, поэтому я вырезал их и объединил в отдельном модуле — `BlnConst.pas`, который мы в дальнейшем будем присоединять ко всем остальным проектам. Этот модуль присутствует на компакт-диске, его также используют и все примеры.

В примере каталога 01_BlnAppMin приложение использует позднее связывание с MS Agent, хотя это вовсе необязательно для применения BalloonDialog. Другой аналогичный пример из каталога 02_BlnAppMin2 построен обычным образом, с визуальным элементом MS Agent. В нем также

реализовано подключение модуля `BlnConst.pas` с константами, благодаря чему вызов BalloonDialog обзавелся значением более информативного стиля выноски:

```
var
 strText: String;
 strTitle: String;
begin
 strText := 'Привет! Это пример минимального приложения.';
 strTitle := 'Привет из BalloonDialog';
 Bln.MsgBalloon(strText, bdInformation, strTitle, Chars);
end;
```

Четвертый и пятый параметры функции `MsgBalloon` указывают ссылку на персонаж, около которого нужно показывать текстовую выноску, эти параметры должны иметь тип `IAgentCtlCharacterEx` и `IAgentCharacterEx` соответственно. Указывать нужно только один из этих параметров. Так, если мы используем Agent Control, то передается только параметр `IAgentCtlCharacterEx`, в случае Agent Server — `IAgentCharacterEx`. Пользуясь поздним связыванием, мы можем пропускать некоторые параметры и попросту их не указывать, как это делается в Visual Basic.

Замечание

Во всех методах, вызывающих появление выноски, — `MsgBalloon`, `InputBalloon`, `ShowFormBalloon`, `Suggest` и `TipBalloon` — для указания ссылки на интерфейс персонажа служат последние два параметра: `oAgent` и `oAgentSvr` типа `IAgentCtlCharacterExIAgentCharacterEx` соотвественно.

Итак, каждый раз при запуске приложения нашему вниманию открывается Trial-окно. Так как конечным пользователям не всегда будет удобно его закрывать, то в этом им нужно помочь. Для этого необходимо программно отыскать нужное окно в системе, а затем послать ему сообщение `WM_CLOSE`. Окно отыскивается по его имени функцией `FindWindow`, которая возвращает дескриптор (описатель) окна. Имя класса можно увидеть с помощью любого менеджера процессов и окон, в данном случае это имя — `ThunderRT6FormDC`. Вот пример программы, помогающей пользователю закрывать Trial-окно (проект каталога 03_TrialHide):

```
program Project1;
uses
  Windows,
  Messages;
var
 HBTN: HWND;
```

```
begin
  HBTN:=FindWindow('ThunderRT6FormDC', nil);
  SendMessage(HBTN, WM_CLOSE, 0, 0);
end.
```

Как вы уже догадались, сначала мы отыскиваем окно и его дескриптор, затем сохраняем в переменную `HBTN` и посылаем этому окну сообщение с помощью функции `SendMessage`.

Теперь, если вы откроете приложение, использующее незарегистрированную версию BalloonDialog, и при открытом информационном окне запустите приложение из каталога 03_TrialHide, то информационное окно закроется и приложение продолжит свою работу.

Вроде бы все понятно. Однако если попытаться реализовать эти функции в одном приложении, то ничего не получится. Вставлять приведенный выше код до создания экземпляра объекта `Balloon` не имеет смысла, т. к. информационное окно еще не существует; с другой стороны, код, добавленный после создания объекта, не выполнится до тех пор, пока не будет закрыто окно, т. к. оно модальное. Единственный выход из данного противоречия состоит в том, чтобы запускать данный код из другого потока (!). Перед запуском функции `CreateOleObject` мы создадим поток, а после завершения работы функции удалим его. Код, в котором реализована эта идея, представлен в проекте каталога 04_TrialWndClose (листинг 4.2).

Листинг 4.2. Модуль главной формы проекта каталога 04_TrialWndClose

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ComObj, BlnConst, OleCtrls, ActiveX, AXCtrls,
AgentObjects_TLB, ExtCtrls;


type
  TBlnHide = class(TThread) //Класс потока
  protected
    procedure Execute; override; // Запуск
    procedure Tic;               // Один тик
  end;

  TForm1 = class(TForm)
    Button1: TButton;
```

```
    Agent1: TAgent;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  end;

var
  Form1: TForm1;
  Chars: IAgentCtlCharacterEx;
  Bln: OleVariant;

  HBTN: HWND;//Дескриптор окна
  BH: TBlnHide;//Объект класса потока

implementation
{$R *.DFM}

procedure TBlnHide.Execute;
begin
 repeat
   synchronize(Tic);
 until Terminated
end;

procedure TBlnHide.Tic;
begin
  HBTN:=FindWindow('ThunderRT6FormDC', nil);
  SendMessage(HBTN, WM_CLOSE, 0, 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Agent1.Characters.Load('Agent', 'merlin.acs');
  Chars:=Agent1.Characters.Character('Agent') as IAgentCtlCharacterEx;
  Chars.Show(0);

  BH:=TBlnHide.Create(false); // Создаем поток
  BH.Resume;
```

```
  Bln:=CreateOleObject('BlnDialogT.Balloon');
  Bln.ResetProperties;

  BH.Terminate;
  BH.Free;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
 Agent1.Characters.UnLoad('Agent');
end;

procedure TForm1.Button1Click(Sender: TObject);
var
 strText: string;
 strTitle: String;
begin
 strText := 'Это пример BalloonDialog, в котором Trial-окно закрывается'+
      'отправкой сообщения из другого потока.';
 strTitle := 'BalloonDialog example';
 Bln.MsgBalloon(strText, bdInformation, strTitle, Chars);
end;
end.
```

В этом случае, как только окно будет закрыто из другого информационного потока, выполнение кода продолжится, и поток будет удален.

Замечание

Иногда при выполнении подобного кода отладчик выдает сообщение "Error in Trial form".

Кроме того, можно подстраховать код другого потока введением цикла, в котором будет реализован постоянный поиск данного окна, прекращающийся лишь тогда, когда окно будет найдено и закрыто:

```
procedure TBlnHide.Tic;
var
 HiBln: Boolean;
begin
 HiBln:=false;
```

```
 repeat
  HBTN:=FindWindow('ThunderRT6FormDC', nil);
  if HBTN<>0 then
   begin
    SendMessage(HBTN, WM_CLOSE, 0, 0);
    HiBln:=true;
   end;
 until HiBln;
end;
```

Этот прием использован в примере каталога 05_TrialWndClose2, однако должен предупредить, в некоторых случаях приложение с подобным кодом при загрузке безнадежно повисает. Выбирайте, что для вас лучше.

Теперь перейдем к подробному рассмотрению свойств и методов BalloonDialog и к описанию всех типов его выносок. Все последующие примеры ориентированы на лицензионную версию BalloonDialog.

Основные свойства и методы BalloonDialog

Перед рассмотрением различных модификаций текстовых выносок познакомимся со свойствами BalloonDialog, изменяющими поведение выноски, а также с теми свойствами, которые пригодятся нам для обеспечения совместимости стиля воздушного шарика с дизайном главного приложения.

Несмотря на то, что каждый элемент управления обладает свойствами, управляющими размерами (шириной и высотой), объект `Balloon` имеет лишь свойство задания ширины в пикселах — `BalloonWidth`. Объясняется это тем, что высоту текстовой выноски не приходится изменять, она регулируется автоматически в зависимости от размера выводимого текста, а заведомо бóльшие или меньшие размеры нет смысла указывать. В большинстве случаев значение даже этого свойства оставляют равным значению по умолчанию.

Далее хотелось бы отметить именно те свойства, с помощью которых можно добиться того, чтобы дизайн BalloonDialog вписывался в дизайн общего приложения. Неполный список этих свойств представлен в табл. 4.2.

Таблица 4.2. *Свойства BalloonDialog*

Свойство	Описание
`BackColor: OLE_COLOR;`	Цвет фона выноски
`BorderColor: OLE_COLOR;`	Цвет бордюра
`ForeColor: OLE_COLOR;`	Цвет текста надписей

Таблица 4.2 *(окончание)*

Свойство	Описание
`LineColor: OLE_COLOR;`	Цвет линии, которая отчеркивает текст от кнопок
`Font: IFontDisp;`	Шрифт текста. Имеет следующие свойства: `Font.Bold: Boolean` — полужирный; `Font.Charset: Integer` — набор символов (для русского языка ничего менять не нужно); `Font.Italic: Boolean` — курсив; `Font.Name: String` — имя шрифта; `Font.Size: Single` — размер шрифта; `Font.Strikethrough: Boolean` — перечеркнутый; `Font.Underline: Boolean` — подчеркнутый; `Font.Weight: Integer` — аналог жирности
`IconPicture: IPictureDisp;`	Значок в выноске
`ButtonPicture: IPictureDisp;`	Изображения вида командных кнопок
`OptionPicture: IPictureDisp;`	Изображения вида кнопок опций (кнопки опций выбора тем в выноске помощника в приложениях MS Office)
`LinkMouseIcon: IPictureDisp;`	Изображение курсора мыши при наведении на кнопки опций
`TexBoxBColor: OLE_COLOR;`	Цвет текстовых полей для ввода текста (в выносках, отображаемых методами `InputBalloon` и `FormBalloon`)
`URLIconPicture: WideString;`	URL-картины для значка в окне Balloon. Обычно применяется, если BalloonDialog используется в web-странице
`URLButtonPicture: WideString;`	URL-картины для командных кнопок
`URLOptionPicture: WideString;`	URL-картины для кнопок опций

Сейчас мы рассмотрим работу только первых семи приведенных свойств на примере уже знакомого нам метода `MsgBalloon`, остальные свойства будут

описаны в ходе обзора методов, вызывающих другие типы выносок[6] (реализация находится в проекте 06_Properties).

Свойства, изменяющие цвета (фона, бордюра и т. п.) и имеющие тип `OLE_COLOR`, используются таким же образом, как обычные свойства типа `TColor`:

```
Bln.BackColor := $829EB2;
```

Вы можете менять параметры шрифта как напрямую через отдельные свойства, так и используя обычный тип `TFont` для преобразования его в `IFontDisp`. Для этого нужно использовать функцию `GetOleFont` из модуля `AXCtrls`.

```
procedure GetOleFont(Font: TFont; var OleFont: IFontDisp);
```

Например:

```
var
 FOLE: IFontDisp;
 ...
begin
 ...
 GetOleFont(Label1.Font, FOLE);
 Bln.Font:=FOLE;
 ...
end;
```

Подобным образом нужно поступать со свойствами, хранящими изображения кнопок и значков, т. к. они имеют тип `IPictureDisp`. Чтобы применять привычные для Delphi-программистов структуры, работающие с картинками (класс `TPicture`), следует использовать функцию `GetOlePicture` для преобразования рисунка к OLE-формату изображения, которая также объявлена в модуле `AXCtrls`.

```
procedure GetOlePicture(Picture: TPicture; var OlePicture: IPictureDisp);
```

Свойство, содержащее изображение командных кнопок, называется `ButtonPicture`, а свойство, определяющее значок в окне Balloon — `IconPicture`. В следующем комплексном примере использованы эти и другие свойства (рабочее окно приложения показано на рис. 4.32):

```
var
 strText: String;
 strTitle: String;
```

[6] Так как увидеть работу части свойств можно только с использованием других типов выносок BalloonDialog, то они будут задействованы при описании других методов.

```
 IPD: IPictureDisp;
 Pic: TPicture;
 FontName: string;
 FontSize: integer;
begin
 strText := 'Пример работы свойств BackColor, BorderColor, ForeColor, '+
     'LineColor, Font, ButtonPicture и IconPicture.';
 strTitle := 'Преимущества BalloonDialog';

 Bln.BackColor := $829EB2;
 Bln.ForeColor := $FFFFFF;
 Bln.LineColor := $60808F;
 Bln.BorderColor := $60808F;

 FontName:=Bln.Font.Name;
 FontSize:=Bln.Font.Size;
 Bln.Font.Name:='Comic Sans MS';
 Bln.Font.Size:=10;

 pic:=TPicture.Create;
 pic.LoadFromFile('..\ButtonCustom.bmp');
 GetOlePicture(pic, IPD);
 Bln.ButtonPicture := IPD;

 pic.LoadFromFile('..\IconCustom3.bmp');
 GetOlePicture(pic, IPD);
 Bln.IconPicture:=IPD;
 pic.free;

 Bln.BorderFocus := False;

 Bln.MsgBalloon(strText, bdCustom, strTitle, Chars);

 Bln.Font.Name:=FontName;
 Bln.Font.Size:=FontSize;
 Bln.ResetProperties;
end;
```

Заметьте, что при использовании другого типа значка при вызове Balloon нужно указывать иной стиль — `bdCustom`.

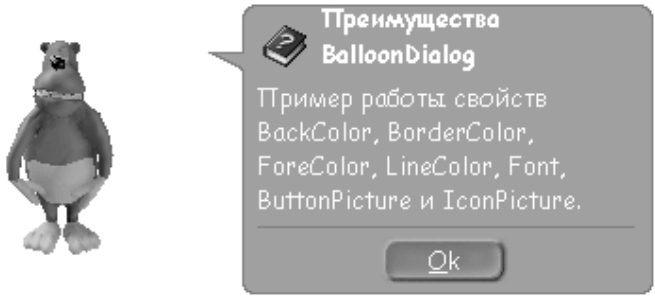


Рис. 4.32. Рабочее окно приложения, демонстрирующего свойства `BackColor`, `Font`, `ButtonPicture` и др.

Замечание

В приведенном выше коде реализовано сохранение текущих параметров свойства `Font`, чтобы после отображения выноски вернуть его в прежнее состояние. Причина этих действий может показаться непонятной, но, как ни странно, методом `ResetProperties` свойство `Font` в значение по умолчанию не возвращается.

Необходимо отметить, что изображения вида кнопок формируются определенным образом, как показано на рис. 4.33. Изображение кнопки обычного вида должно быть самым верхним, под ним — изображение кнопки в том виде, который она принимает, когда указатель мыши находится над ней, внизу — изображение нажатой кнопки. Размер кнопки может быть различным, но кадры различных ее видов обязаны иметь одинаковый размер.

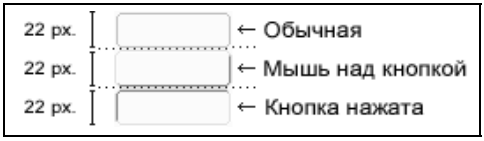


Рис. 4.33. Схема расположения изображений кнопок трех видов в файле картинки для свойства `ButtonPicture`

Кнопка, на которой в данный момент установлен фокус, в выноске обводится характерной пунктирной линией, как на обычных кнопках, однако этой особенностью также можно управлять. Свойство логического типа `BorderFocus` позволяет определить, нужно ли выводить пунктирную линию,

если кнопка в фокусе. Так, например, по умолчанию при использовании кнопок с изображениями данное свойство автоматически отключается. Если это отключение нежелательно, то после определения кнопочной картинки нужно включить свойство `BorderFocus`.

По умолчанию любая выноска Balloon автоматически перемещается к указанному при вызове персонажу. Регулируется этот процесс с помощью свойства `MoveStyle`, задающего стиль перемещения. Свойство может иметь следующие допустимые значения: 0 — стиль MoveManual, 1 — MoveHiding и 2 — MoveFollowing. При значении 0 пользователь может переместить выноску только вручную посредством мыши. Значение 2 (MoveFollowing) является значением по умолчанию. Действие, определяемое значением 1, схоже с действием MoveManual, т. е. выноска не передвигается автоматически, для ее перемещения нужно использовать комбинацию методов `HideBalloonToMove`, `SetCharacterPosition` и `MoveBalloonToCharacter`. Подобным образом реализовано перемещение "воздушных шариков" за персонажем в MS Office. В начале перемещения персонажа выноска скрывается, а при окончании появляется в нужном месте около него.

Замечание

В документации к BalloonDialog указывается, что значение, определяющее стиль MoveHiding, не работает в Visual Basic и Visual Fox Pro (это замечание я привожу в качестве цитаты, т. к. нас данный факт не должен беспокоить).

Перейдем к рассмотрению методов, управляющих процессом перемещения выноски.

Метод `HideBalloonToMove` скрывает активный Balloon для последующего перемещения, поэтому у него отсутствуют какие-либо параметры. Метод `SetCharacterPosition` используется для того, чтобы задать параметры расположения персонажа (либо какого-то другого объекта), около которого должна появиться выноска — это координаты X, Y и ширина кадра персонажа. Метод `MoveBalloonToCharacter`, также не имеющий параметров, отображает выноску после перемещения ее посредством мыши.

Замечание

Используя стиль перемещения 1 (MoveHiding), при вызове соответствующей выноски можно не указывать параметры `oAgent` или `oAgentSvr`, т. е. последние два параметра в методах `MsgBalloon`, `InputBalloon`, `ShowFormBalloon`, `Suggest` и `TipBalloon`.

Обычно стиль MoveHiding используется следующим образом:

```
procedure TForm1.Agent1DragStart(Sender: TObject;
  const CharacterID: WideString; Button, Shift, x, y: Smallint);
```

```
begin
  Bln.HideBalloonToMove;

procedure TForm1.Agent1Move(Sender: TObject;
                      const CharacterID: WideString;
  x, y, Cause: Smallint);
begin
  Bln.SetCharacterPosition(Chars.Left, Chars.Top, Chars.Width);
  Bln.MoveBalloonToCharacter;
end;
```

В данном случае даже не имеет смысла устанавливать условие `Bln.MoveStyle = 1`, т. к. при любом другом стиле перемещения эти методы не работают.

В случае ручного перемещения выноски пользователь может двигать ее посредством мыши, однако и здесь бедного "юзера" можно ограничить, если того требует ваш проектный замысел. Свойство логического типа `UserMove` позволяет определить, разрешено ли такое ограниченное перемещение; по умолчанию свойство включено, т. е. имеет значение `True`, что дает возможность допустить передвижение шарика. Это может потребоваться, если выноской нужно указать на какой-либо элемент интерфейса — например так, как в Windows XP система показывает недавно установленные программы, изменения в системном лотке и т. п.

Поиграв с различными вариантами положения текстовой выноски около персонажа, можно заметить, что всякий раз угол выноски отображается на границе начала кадра персонажа. Для регулирования этой дистанции нужно воспользоваться парой целочисленных свойств `BalloonDistanceX` и `BalloonDistanceY`. Они хранят количество пикселов, на которое удален воздушный шарик от персонажа. Значения свойств могут быть как положительными, так и отрицательными. В случае положительных величин выноска располагается дальше от границы кадра агента, при отрицательных — ближе.

В проекте каталога 06_Properties реализована демонстрация рассмотренных свойств. С остальными свойствами мы познакомимся при рассмотрении других типов выносок, т. к. некоторые из них имеют смысл только в сочетании с еще не рассмотренными нами типами Balloon, к описанию которых мы сейчас и переходим.

Метод *MsgBalloon*

Минимальные примеры с BalloonDialog были построены с применением метода `MsgBalloon`. Комбинируя те или иные параметры, мы можем добиваться различных видов воздушных шариков. Напомню прототип метода:

```
function  MsgBalloon(const Prompt: WideString; Buttons: BalloonStyle;
const Title: WideString; const oAgent: IAgentCtlCharacterEx; const
oAgentSvr: IAgentCharacterEx): MsgBalloonResult;
```

С помощью данного метода можно строить информационные окна как минимум с одной кнопкой **Ok**, без значков и заголовков, а как максимум — заменить стандартные диалоговые сообщения, вызываемые функцией `MessageDlg`.

Параметр `Prompt` — текст в окне, может достигать длины в 1024 символа. Если не указать значение параметра `Title`, то в выноске будет отсутствовать заголовок, а пропустив параметр, задающий стиль (`Buttons`), мы тем самым определим отсутствие значка в окне, а также наличие по умолчанию одной кнопки **Ok**. Для того чтобы использовать все возможности метода `MsgBalloon`, нужно изменять второй параметр этого метода. Этот параметр называется `Buttons`, он включает в себя как набор кнопок для отображения, так и тип пиктограммы в сообщении. Объединяются эти значения через сумму, как в Visual Basic в функции `MsgBox`. Возможные значения можно просмотреть в модуле констант, они также приведены в табл. 4.3.

Таблица 4.3. *Константы стилей для диалоговых окон*

Константа	Значение	Описание
	Константы типа "кнопка"	
bdOKOnly	$00000000	Только **Ok**
bdOKCancel	$00000001	**Ok** и **Cancel**
bdAbortRetryIgnore	$00000002	**Abort**, **Retry** и **Ignore**
bdYesNoCancel	$00000003	**Yes**, **No** и **Cancel**
bdYesNo	$00000004	**Yes** и **No**
bdRetryCancel	$00000005	**Retry** и **Cancel**
	Константы типа "значок"	
bdCritical	$00000010	Ошибка
bdQuestion	$00000020	Вопросительный знак
bdExclamation	$00000030	Восклицательный знак
bdInformation	$00000040	Информационное сообщение

Таблица 4.3 *(окончание)*

Константа	Значение	Описание
	Константы типа "значок"	
bdCustom	$00000050	Выборочный значок, указываемый через свойство IconPicture или URLIconPicture

Указав в параметре Buttons сумму двух нужных констант, можно добиться вывода нужных кнопок и значков, например:

```
Bln.MsgBalloon(strText, bdOKCancel + bdExclamation, strTitle, Chars);
```

Обратите внимание, что выбирать необходимо из имеющихся констант, а не составлять экзотических наборов кнопок или значков, например:

```
Bln.MsgBalloon(strText, bdOKCancel + bdYesNo + bdExclamation, strTitle,
Chars);
```

С помощью свойства ButtonsCaptions элемента управления BalloonDialog можно заменять стандартные надписи на командных кнопках на свои собственные. Свойство представляет собой строку с перечисленными через точку с запятой заголовками кнопок. Перед буквой, которая в сочетании с клавишей <Alt> образует быструю комбинацию, ставится знак "&" (амперсанд):

```
Bln.ButtonsCaptions:='&Ok;&Cancel;&Abort;&Retry;&Ignore;&Yes;&No';
```

Следующий пример использует свойство ButtonsCaptions:

```
var
 Result: Integer;
 strText: String;
 strTitle: String;
begin
  strText := 'Вы можете изменять надписи на командных кнопках.';
  strTitle := 'Преимущества BalloonDialog';
  //Изменяем кнопки
  Bln.ButtonsCaptions:='&Понятно;&Отмена;&Прервать;'+
         '&Еще раз;&Плевать;&Угу;&Нет';

  Bln.MsgBalloon(strText, bdOKCancel + bdExclamation, strTitle, Chars);
  Bln.ResetProperties;
end;
```

На рис. 4.34 дан пример эффектов, которых можно достичь с помощью свойств `IconPicture` и `ButtonsCaptions`

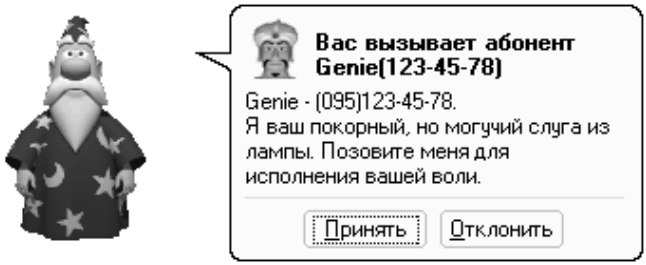


Рис. 4.34. Пример эффектов, которых можно достичь с помощью свойств `IconPicture` и `ButtonsCaptions`

Возвращать надписи в значения по умолчанию можно через метод `ResetProperties` или с помощью прямого присваивания через центральный интерфейс первоначальных значений.

Замечание

Изменения командных кнопочных надписей происходят во всех типах выносок, т. е. в типах, вызываемых методами `MsgBalloon`, `InputBalloon` и `TipBalloon`.

Конечно же, если вы планируете выдавать сообщения на русском языке, то и кнопки хорошо бы "облагородить", однако следует следить, чтобы случайный вызов сброса свойств через метод `ResetProperties` не изменил их на первоначальные.

Так как метод `MsgBalloon` реализован для получения результата от пользователя, то он возвращает значение, по которому можно определить нажатую пользователем кнопку. Возвращаемый результат может иметь несколько значений — они представлены в табл. 4.4.

Таблица 4.4. *Константы, определяющие нажатую кнопку в диалоге*

Константа	Значение	Кнопка
`bdOK`	$00000001	**Ok**
`bdCancel`	$00000002	**Cancel**
`bdAbort`	$00000003	**Abort**
`bdRetry`	$00000004	**Retry**
`bdIgnore`	$00000005	**Ignore**

Таблица 4.4 *(окончание)*

Константа	Значение	Кнопка
bdYes	$00000006	**Yes**
bdNo	$00000007	**No**

Пример кода с определением нажатой пользователем кнопки:

```
var
 Result: Integer;
 strText: String;
 strTitle: String;
begin
  strTitle:= 'Пример BalloonDialog.';
  strText:= 'Вам наравится BalloonDialog?';
  Result := Bln.MsgBalloon(strText, bdYesNoCancel + bdQuestion, strTitle,
Chars);

  Case Result of
   {Yes}6: ShowMessage('Спасибо.');
   {No}7: ShowMessage('Напишите лучше.');
   {Cnacel}2: ShowMessage('У вас еще есть возможность определиться.');
  end;
end;
```

Естественно, вместо прямых значений при сравнении можно использовать как заранее объявленные константы, так и константы из модуля `BlnConst.pas`. Метод `MsgBalloon` продемонстрирован в проекте каталога 07_MsgBalloon.

Метод *InputBalloon*

Метод `InputBalloon` позволяет отобразить окно выноски, подобное тому, какое вызывается функциией `InputBox` и другими подобными функциями. Прототип метода:

```
function  InputBalloon(const Prompt: WideString; const Title: WideString;
      const Default: WideString; const oAgent: IAgentCtlCharacterEx;
      const oAgentSvr: IAgentCharacterEx): WideString;
```

Первый параметр — строка приглашения или вводный текст, может достигать длины в 1024 символов, второй — заголовок окна, третий — значение

по умолчанию, последние два параметра служат для указания на интерфейс персонажа.

Пример кода с использованием этого метода:

```
var
 sResult: String;
begin
  Bln.ButtonsCaptions := '&Ок;&Отмена';
 sResult := Bln.InputBalloon('Пожалуйста, введите ваше имя.', 'Как вас
зовут?', , Chars);
 if sResult = '' Then
  exit;
 ShowMessage('Привет, '+sResult+'!');
end;
```

Код отображает выноску, похожую на ту, что вы видите на рис. 4.35.

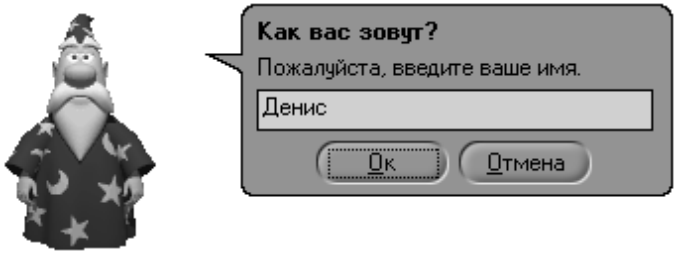


Рис. 4.35. Пример выноски, полученной с помощью метода `InputBalloon`

Мы уже говорили о свойствах компонента BalloonDialog, которые позволяют текстовой выноске вписаться в дизайн вашего приложения, так вот для этих целей используется еще одно свойство — `TexBoxBColor`, позволяющее изменять цвет текстового поля окна. Этот эффект отражен на рис. 4.35.

Метод `InputBalloon` продемонстрирован в проекте каталога 08_InputBalloon.

Метод *TipBalloon*

Метод `TipBalloon` позволяет отображать выноски в стиле подсказок, не имеющих фокуса и закрывающихся при щелчке мышью. Такие же выноски встречаются в Microsoft Office или Windows XP в качестве упрощенных информационных окон — характерно вытянутых по горизонтали.

Прототип метода:

```
procedure  TipBalloon(const Tip: WideString; IDTip: Smallint; Button:
WordBool; const ChkBoxText: WideString; TimeOut: Single; TipWidth:
Smallint; const oAgent: IAgentCtlCharacterEx; const oAgentSvr:
IAgentCharacterEx);
```

Первый (обязательный) параметр хранит текст подсказки длиной не более 1024 символов. Второй параметр можно использовать как идентификатор для подсказки, который передается в событии `OnTipClose`. Логический параметр `Button` показывает, нужно ли отображать кнопку **Ok** в окне выноски, причем кнопка будет иметь надпись, которая указана в свойстве `ButtonsCaptions`.

Если в воздушном шарике нужно отобразить флажок с определенным текстом, то следует задать следующий параметр: `ChkBoxText`. Если пропустить его, выноска отобразится без флажка. Если флажок был отображен, то его состояние также можно будет узнать в событии `OnTipClose`. Должен сказать, что рассчитывать на обработку событий компонентом BalloonDialog не стоит, т. к. автору не удалось в позднем связывании заставить Delphi обработать события этого компонента. Конечно, если кому-то из читателей удастся добиться этого или заставить BalloonDialog работать в Delphi не только в позднем связывании, то ему и карты в руки. Пока же мы познакомимся с событиями лишь теоретически, т. к. в интерпретируемых языках, таких как Visual Basic и VBScript, события работают.

Параметром `TimeOut` указывается промежуток времени в миллисекундах, по истечении которого выноска будет автоматически скрываться. Максимальное значение параметра — 65 535, что составляет чуть более одной минуты.

С помощью параметра `TipWidth` определяется ширина окна Balloon, благодаря чему можно добиться вытягивания размеров окна (см. рис. 4.36).

Последние два параметра предназначены, как всегда, для передачи указателя на нужные интерфейсы персонажа.

Следующий код отображает простейшую подсказку в том виде, как она реализована в MS Office:

```
var
 strTip: String;
begin
  strTip := 'Этот TipBalloon не имеет фокуса. '+
      'Для скрытия выноски щелкните на ней...';
  Bln.TipBalloon(strTip, 1, False, , , 150, Chars);
end;
```

Следующий код отображает Balloon в стиле горизонтально вытянутых выносок Windows XP:

```
var
 strTip: String;
begin
 strTip := 'Это часы.'+#10#13+#10#13+
     'Вы должны использовать метод SetManualPosition, '+
     'чтобы изменять координаты X и Y стрелки.';
 //Указываем, что выноску перемещаем вручную
 Bln.MoveStyle := bdMoveManual;
 //Координаты в пикселах
 Bln.SetManualPosition(Screen.Width, Screen.Height-30);
 //Показываем выноску
 Bln.TipBalloon(strTip, 2, false, , 8000, 260);
 Bln.ResetProperties
end;
```

Примеры изображений приведены на рис. 4.36.

Рис. 4.36. Примеры выносок, полученных с помощью метода `TipBalloon`

Примеры, описанные в данном подразделе, содержит проект из каталога 09_TipBalloon.

Метод *Suggest*

Метод `Suggest` позволяет создать Balloon любой формы на основе картинки (см. рис. 4.37). Прототип метода:

```
procedure  Suggest(IDSuggest: Smallint; TimeOut: Single; const oAgent:
IAgentCtlCharacterEx; const oAgentSvr: IAgentCharacterEx);
```

Первый параметр определяет идентификатор выноски аналогично параметру `IDTip` в методе `TipBalloon`; он может пригодиться в событии `OnSuggestClick`. Второй параметр определяет тайм-аут, в течение которого выноска ждет щелчка мышью. Последние два параметра аналогичны одноименным параметрам предыдущих методов.

Следующий код показывает один из возможных вариантов запуска метода `Suggest`:

```
begin
 Bln.SetCharacterPosition(Chars.Left, Chars.Top, Chars.Width);
 Bln.Suggest(1, , Chars);
end;
```

По умолчанию изображение выноски представляет собой лампочку, похожую на ту, которая имеется в приложениях MS Office. Для того чтобы сменить картинку, необходимо пользоваться свойством `SuggestPicture` типа `IPictureDisp`, определяющим картинку, и свойством `MaskColor` типа `OLE_COLOR` для указания цвета прозрачности в изображении `SuggestPicture`.

```
var
 IPD: IPictureDisp;
 pic: TPicture;
begin
 pic:=TPicture.Create;
 pic.LoadFromFile('..\Clock.bmp');
 GetOlePicture(pic, IPD);
 pic.free;
 Bln.SuggestPicture := IPD;
 Bln.Suggest(2, , Chars);
end;
```

Примеры выносок, созданных на основе картинок, даны на рис. 4.37.

Смотрите также демонстрационный проект из каталога 10_Suggest.

Рис. 4.37. Примеры выносок, созданных с помощью метода `Suggest`

Объект *FormBalloon* и метод *ShowFormBalloon*

Через объект `FormBalloon` настраиваются параметры выноски, отображаемой в виде многофункциональной формы, а методом `ShowFormBalloon` запускается выноска в стиле информационного окна MS Office, через которое пользователь задает ключевые слова для поиска в справочной системе, а также выбирает опции. Данный воздушный шарик, пожалуй, самый "интерфейсный" и многофункциональный.

Объект `FormBalloon` имеет следующие свойства, представленные в табл. 4.5.

Таблица 4.5. *Свойства объекта* `FormBalloon`

Свойство	Описание
`Icon: IconConstants`	Тип значка для отображения *(см. ниже)*
`Title: WideString`	Заголовок окна
`Message: WideString`	Текст сообщения длиной до 1024 символов
`OptionButtons: _cclsOptButtons`	Указатель на интерфейс для работы с кнопками опций *(см. ниже)*
`TextBox: WordBool`	Определяет, нужно ли выводить текстовое поле для ввода информации
`TBoxLines: Smallint`	Определяет количество строк в текстовом поле. Значение по умолчанию 2
`TBoxText: WideString`	Текст в текстовом редакторе
`SepLine: WordBool`	Определяет, нужно ли выводить разделительную линию между комментарием и командными кнопками

Таблица 4.5 *(окончание)*

Свойство	Описание
`Comment: WideString`	Комментарий в нижней части окна выноски
`Buttons: _cclsButtons`	Указатель на интерфейс для работы с командными кнопками *(см. ниже)*
`ShowStyle: ShowStyleConstants`	Стиль вывода выноски: модальный или немодальный *(см. ниже)*
`OptionPressed: Smallint`	Номер нажатой опциональной кнопки после вывода выноски
`ButtonPressed: Smallint`	Номер нажатой командной кнопки после вывода выноски

В табл. 4.5 указан полный перечень свойств, которые можно использовать при создании подобной выноски. Естественно, что части из них можно не присваивать значения, получая, таким образом, окно без заголовка или без значка, и т. п.

Значки в окне `FormBalloon` имеют такие же стили, как в окне `MsgBalloon`, однако соответствующие им значения констант отличаются, поэтому привожу их в табл. 4.6.

Таблица 4.6. *Константы значков для окна* `FormBalloon`

Константа	Значение	Описание
`bdNone`	$00000000	Без значка
`bdIconCritical`	$0000000B	Значок ошибки
`bdIconQuestion`	$0000000C	Значок с вопросительным знаком
`bdIconExclamation`	$0000000D	Восклицательный знак
`bdIconInformation`	$0000000E	Значок информации
`bdIconCustom`	$00000050	Произвольный значок. Загружается через свойства `IconPicture` или `URLIconPicture` элемента управления

Изображения для кнопок опций загружаются через свойство `OptionPicture` таким же способом, как и картинки в свойстве `ButtonPicture`. Они могут быть произвольного размера, однако нужно следить за совпадением размеров различных кадров изображений. Кнопки опций могут находиться, кроме трех состояний, свойственных командным кнопкам (`Normal`, `MouseOver`,

MouseDown), в состоянии Disabled. Располагаться они должны так, как показано на рис. 4.38.

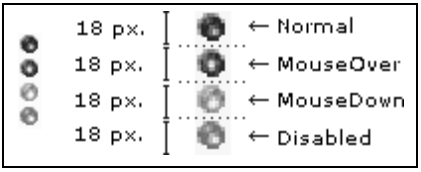


Рис. 4.38. Расположение кадров кнопок опций

Для работы с кнопками опций используется свойство OptionButtons, а для командных — свойство Buttons. Оба свойства являются коллекциями и имеют ряд методов для управления ими. Методы и свойства коллекции OptionButtons приведены в табл. 4.7.

Таблица 4.7. *Методы коллекции* `OptionButtons`

Метод	Описание
`function Add (var Caption: WideString; var TextColor: Integer; var Enabled: WordBool; var sKey: WideString): _clsOptButton;`	Добавляет одну опциональную кнопку. Параметры: Caption — текст надписи; TextColor — цвет текста (например, для выделения посещенных ссылок); Enabled — определяет, доступна ли опция; sKey — уникальная текстовая строка, которая может использоваться вместо порядкового индекса для обращения к элементу коллекции
`procedure Remove (var vIndexKey: OleVariant);`	Удаляет кнопку из коллекции по индексу или ключу
`property Item[var vIndexKey: OleVariant]: _clsOptButton`	Список коллекции для обращения к каждому элементу (элементы нумеруются с 1)
`property Count: Integer`	Показывает количество кнопок в коллекции

Каждая опциональная кнопка, т. е. элемент коллекции (интерфейс _clsOptButton) имеет следующие свойства, назначение которых вполне очевидно:

- `property Caption: WideString;`
- `property TextColor: Integer;`
- `property Enabled: Smallint;`

Методы коллекции командных кнопок `Buttons` отличаются друг от друга только параметрами метода (функции) `Add`.

```
function  Add(var Caption: WideString; var sKey: WideString): _clsButton;
```

Функция `Add` также добавляет одну кнопку с надписью `Caption`. В виде параметра `sKey` можно передавать уникальную строку, которой можно пользоваться вместо порядкового номера для обращения к элементу коллекции.

Каждая командная кнопка имеет одно свойство — `Caption` — надпись на кнопке. Обращаться к нему нужно через свойство `Item` коллекции `OptionButton`, которое содержит список указателей на интерфейс `_clsButton` каждой имеющейся кнопки.

После того как все опции настроены, можно отображать форму выноски с помощью метода `ShowFormBalloon`, в котором указывается лишь ссылка на интерфейс персонажа.

```
begin
  Bln.FormBalloon.Buttons.Add('&Угу');
  Bln.FormBalloon.Buttons.Add('&Да ну');

  Bln.FormBalloon.Icon := bdIconExclamation;
  Bln.FormBalloon.Title := 'Что может BalloonDialog?';
  Bln.FormBalloon.Message := 'Выберите то, о чем вы желаете узнать или'+
    ' добавьте свой вопрос.';

  Bln.FormBalloon.OptionButtons.Add('&О функциях BalloonDialog');
  Bln.FormBalloon.OptionButtons.Add('&Программирование с BalloonDialog');
  Bln.FormBalloon.OptionButtons.Add('&Возможные исключительные '+
       'ситуации');

  Bln.FormBalloon.TextBox:=True;
  Bln.FormBalloon.TBoxText:='Ваш вопрос';

  Bln.FormBalloon.Comment:='Запишите сюда ваш вопрос, '+
         'если его нет в списке.';

  Bln.FormBalloon.SepLine:= True;
  //Показываем выноску
  Bln.ShowFormBalloon(Chars);
  //Поулчаем результат
  if Bln.FormBalloon.OptionPressed = 0 then
```

```
    ShowMessage('Вы нажали кнопку под номером: '
        +IntToStr(Bln.FormBalloon.ButtonPressed))
  else
    ShowMessage('Вы выбрали опцию под номером: '
        +IntToStr(Bln.FormBalloon.OptionPressed));
  ShowMessage('Ваш текст: '+Bln.FormBalloon.TBoxText);
end;
```

Данный код создает окно, показанное на рис. 4.39.

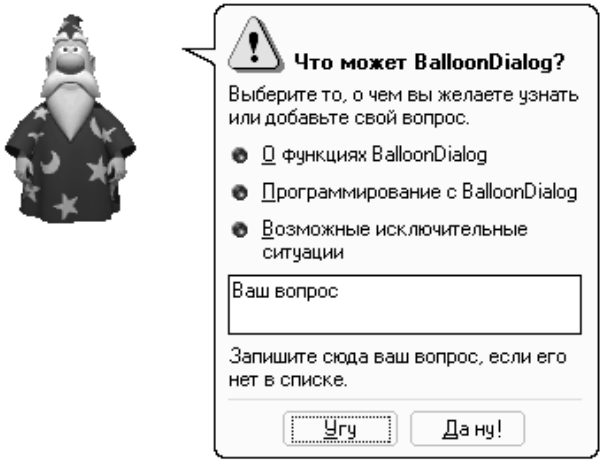


Рис. 4.39. Пример выноски, созданной с помощью объекта `FormBalloon`

После запуска выноски можно оценить пользовательский ответ и введенную информацию. Для этого как раз и нужны свойства `OptionPressed` и `ButtonPressed`. При выборе кнопки опций либо командных кнопок соответствующее свойство будет содержать ненулевое значение, по которому можно будет определить порядковый номер кнопки в коллекции, которая была нажата. Свойство `FormBalloon.TboxText` будет содержать текст, введенный пользователем.

Проект каталога 11_FormBalloon содержит несколько ярких примеров использования этого типа выноски BalloonDialog. Все они показаны на рис. 4.40.

Надеюсь, читателям понравился рассказ о замечательном компоненте BalloonDialog, который отличается высокой функциональностью и большим сродством с анимированным персонажем Microsoft Agent.

Рис. 4.40. Примеры использования выноски `FormBalloon`

4.5. Совмещаем технологии MS Agent и Speech API

И в заключение хотелось бы рассказать о возможности объединения технологий MS Agent и Speech API. Как известно, MS Agent самостоятельно не синтезирует и не распознает речь, он использует для этого функции Speech API.

Синтез речи с помощью методов MS Agent построен удобно, как и распознавание команд. Однако в проектах, например, *гл. 2* для большей полноты функций желательно сделать выбор движков из набора реально установленных в системе, как это делается в примерах *гл. 3*. Методами MS Agent такой выбор невозможно реализовать, поэтому нужно пользоваться средствами технологии Speech API.

В тех приложениях, где необходим выбор TTS-движка, надо использовать интерфейс `ITTSEnum` объекта Text-To-Speech Enumerator. Перебор движков осуществляется точно так же, как в приложениях Speech API. При выборе

движка из списка нужно определить его идентификатор `ModeID` и присвоить его свойству `TTSModeID`, например:

```
var
  aTTEnum: ITTEnum;
  fttsModeInfo : TTSModeInfo;
begin
  CoCreateInstance(CLSID_TTSEnumerator, Nil, CLSCTX_ALL,
    IID_ITTSEnum, aTTSEnum);
  aTTSEnum.Reset;
  aTTSEnum.skip(cbTTSEngine.ItemIndex);
  aTTSEnum.Next(1, fttsModeInfo, @NumFound);
  try
    Chars.LanguageID:=fttsModeInfo.Language.LanguageID;
    Chars.TTSModeID:=GUIDToString(fttsModeInfo.gModeID);
  except
  end;
end;
```

Все вышесказанное касается и модулей распознавания речи. Только здесь применяется интерфейс `ISREnum` объекта Speech Recognition Enumerator:

```
var
  aTTSEnum: ITTSEnum;
  fsrModeInfo: SRModeInfo;
begin
  CoCreateInstance(CLSID_SREnumerator, Nil, CLSCTX_ALL,
    IID_ISREnum, aSREnum);
  aSREnum.Reset;
  aSREnum.Skip(cbSREngine.ItemIndex);
  aSREnum.Next(1, fsrModeInfo, @NumFound);
  try
    Chars.LanguageID:=fsrModeInfo.Language.LanguageID;
    Chars.SRModeID:=GUIDToString(fsrModeInfo.gModeID);
  except
  end;
end;
```

Для получения диапазонов скорости и тона речи TTS-движков используется интерфейс атрибутов синтеза речи (`ITTSAttributes`) и соответствующие ин-

терфейсы Engine Object. Разрабатывая приложения с MS Agent и Speech API с реализацией всех их функций, рекомендуется просто создать полноценное приложение со Speech API, но запускать чтение текста не через метод `TextData` центрального интерфейса `ITTSCentral`, а через метод `Speak` объекта `Character`. Способ несколько громоздкий, т. к. в нем не будут работать интерфейс `ITTSCentral` и связи с аудиоустройством. Ну что ж, зато попутно можно реализовать чтение в файл. В результате совмещения этих функций получится довольно симпатичное приложение. Сюда же можно включить BalloonDialog и другие "красивости". Подобный проект реализован (рис. 4.41) в каталоге 4_MSAgent&SpeechAPI — правда, он еще далек от совершенства. Неплохо добавить в него сохранение настроек в ini-файле и другой косметический ремонт.

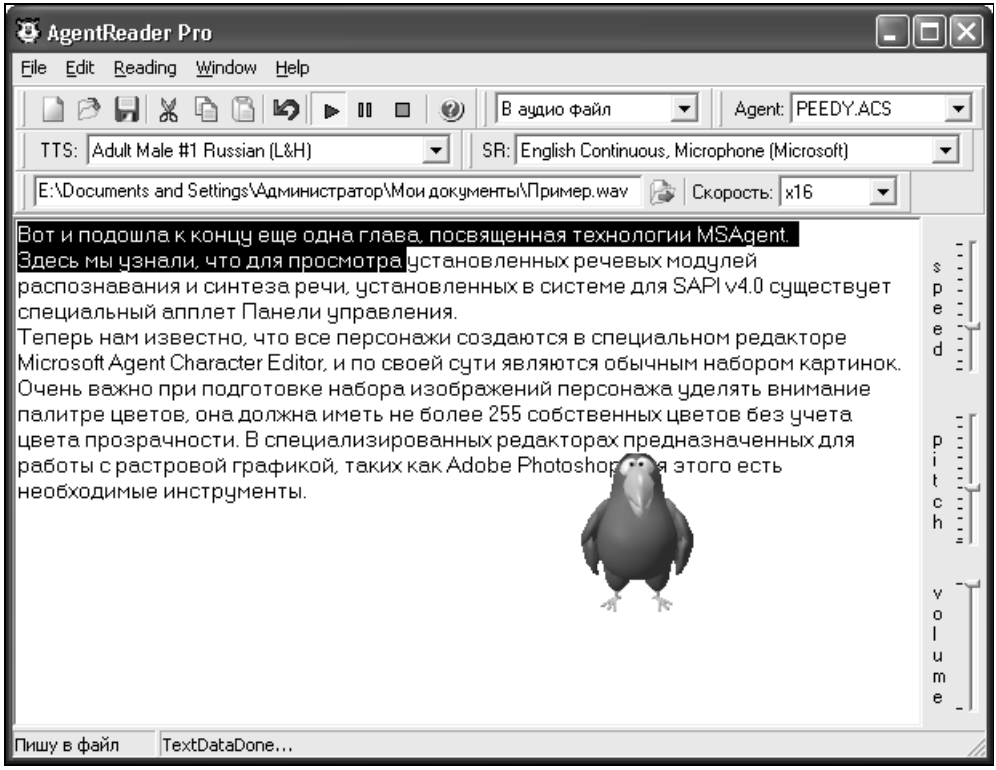


Рис. 4.41. Рабочее окно приложения MS Agent + Speech API + BalloonDialog

Возможно, вы столкнетесь с проблемой, когда после выбора русскоязычного модуля синтеза речи персонаж откажется распознавать английскую речь через английский же движок. В самом деле, объект `Character` имеет одно

свойство LanguageID, управляющее языком, поэтому при смене движка оно приобретает язык последнего выбранного модуля. Чтобы таких проблем не возникало, нужно всего лишь вовремя менять язык помощника на язык модуля синтеза или распознавания речи. Например, следующим образом:

```
try
  Chars.LanguageID:=fttsModeInfo.Language.LanguageID;
  Chars.TTSModeID:=GUIDToString(fttsModeInfo.gModeID);
except
end;
```

или

```
try
  Chars.LanguageID:=fsrModeInfo.Language.LanguageID;
  Chars.SRModeID:=GUIDToString(fsrModeInfo.gModeID);
except
end;
```

Полный код приложения, реализующего синтез и распознавание речи с поддержкой MS Agent, смотрите в каталоге 4_MSAgent&SpeechAPI.

4.6. Резюме

Вот и подошла к концу последняя глава, посвященная технологии MS Agent.

Здесь мы узнали, что для просмотра модулей распознавания и синтеза речи, установленных в системе, для SAPI v4.0 существует специальный апплет Панели управления.

Теперь нам известно, что все персонажи создаются в специальном редакторе Microsoft Agent Character Editor и являются по своей сути обычным набором картинок. Очень важно при подготовке набора изображений персонажа уделять внимание палитре цветов, которая должна иметь не более 255 собственных цветов без учета цвета прозрачности. В специализированных редакторах, предназначенных для работы с растровой графикой, таких как Adobe Photoshop, для этого есть необходимые инструменты.

Персонаж состоит из данных, являющихся его настройками, и собственно анимаций. В персонаже может использоваться множество мультипликаций, все они состоят из кадров — фреймов. В свою очередь, каждый кадр может включать несколько изображений, из которых складывается комплексная иллюстрация. Плюс ко всему на кадр может накладываться специальная анимация губ персонажа, однако она внедряется только в последний кадр мультипликации, т. к. именно во время отображения последнего кадра (т. е. когда анимация завершена) персонаж может выводить речь.

Для серии кадров в анимации персонажа реализованы вероятностные переходы на различные номера кадров. Количество ветвлений с каждого кадра может изменяться от 0 до 3. С помощью этого приема достигается эффект случайного поведения агента — например, во время бездействия. Этот принцип используется также для построения циклических анимаций.

С целью воссоздания плавности переходов между различными анимациями для целевой мультипликации создают возвратные анимации (Return animation), которые воспроизводятся перед выполнением любой следующей анимации, после целевой. Кроме того, для построения выходного ряда кадров используют заключительные переходы **Exit Branching**.

В третьем разделе данной главы мы познакомились со специальным лингвистическим редактором для Microsoft Agent, в котором подготавливаются волновые файлы с внедренной лингвистической информацией. Эта информация помогает персонажу при воспроизведении конечного файла методом `Speak` адекватно шевелить губами.

Наконец, мы освоили такой эффектный компонент, как BalloonDialog от фирмы SommyTech. Он позволяет построить элегантный и более эстетически выигрышный интерфейс пользователя для MS Agent. Можно сказать, что это подарок для разработчиков, использующих Microsoft Agent. Даже некоторое количество исключений и трудностей, возникающих при эксплуатации данного компонента в Delphi, которые начинаются практически с момента его установки, не затмевают его очевидных преимуществ.

Полагаю, все дополнительные компоненты и модули, так называемые "примочки", к MS Agent и Speech API — а в главе была описана лишь небольшая, но самая основная их часть, — весьма понравились читателям. Думаю, в Интернете пытливый программист найдет немало подобных модулей, а может быть, со временем даже напишет свой собственный эксклюзивный компонент.

Заключение

Надеюсь, что в данной книге достаточно полно раскрыт основной круг вопросов по созданию приложений с использованием технологии анимированного персонажа MS Agent и речевых технологий Speech API в Delphi.

Мне кажется, что главу 2 можно рассматривать как исчерпывающую документацию по Microsoft Agent, т. к. в ней описаны все свойства и параметры этого элемента управления, а большое количество приведенных в ней примеров и комментариев к ним помогут разобраться в технологии MS Agent разработчику любого уровня.

В главе 3, посвященной Speech API, описаны все интерфейсы этой технологии, за исключением Voice Telephony API (в силу редкого применения последнего нашими пользователями описание опущено). Эту главу можно считать первым наиболее полным руководством на русском языке по данной теме. Разумеется, количество примеров и рассмотренных нюансов могло быть бóльшим, но описать абсолютно все в одной книге, пожалуй, невозможно. Однако она предоставляет хороший базовый материал для изучения речевых технологий. В главе 3 синтез речи рассмотрен более подробно, нежели ее распознавание, и это неудивительно, т. к. синтез речи применяется намного чаще, чем распознавание. Тем более что, поняв принцип работы синтеза речи, читатель легко разберется и в распознавании. При этом я не имею в виду, что две эти технологии похожи друг на друга как две капли воды, а говорю о сходстве принципов программирования в SAPI.

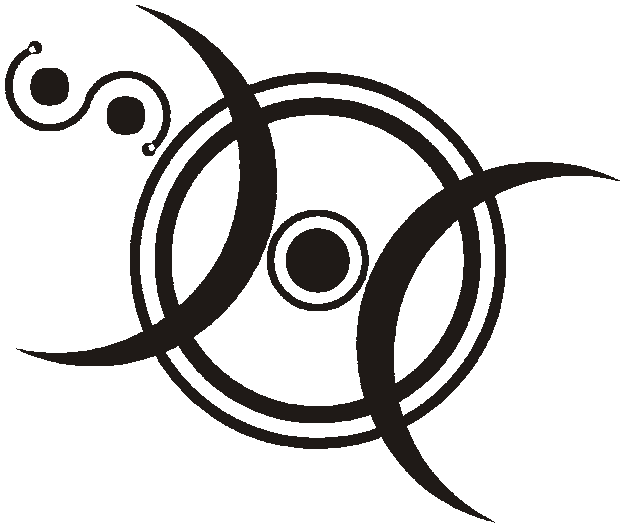
Описание дополнительных компонентов, данное в главе 4, и, в особенности, приведенные в книге примеры создания собственных персонажей должны помочь программистам в применении анимированного персонажа в собственных приложениях.

Конечно, не во всяком приложении можно использовать анимированного агента, но согласитесь, что умеренное применение данных функций способствует приданию приложению более привлекательного вида. Все-таки появление персонажа MS Agent на экране монитора вызовет у пользователей скорее легкую улыбку, чем бурю негодования.

Думаю, что моя книга поможет вам открыть для себя мир удивительных технологий MS Agent и Speech API.

С уважением,
Денис Буторин

E-mail: subritto@ngs.ru
Web: http://subritto.h1.ru



Приложения

Приложение 1

Ссылки на сайты и файлы в Интернете

В данном приложении приведены ссылки на сайты, связанные с MS Agent и речевыми технологиями, а также ссылки на необходимые компоненты, которые можно скачать в Интернете.

Ссылки на сайты

- **http://www.microsoft.com/MS Agent** — сайт, с которого нужно начать изучение MS Agent.
- **http://www.microsoft.com/speech** — сайт, с которого целесообразно начать изучение технологии Speech API.
- **http://bdn.borland.com** — полагаю, что этот сайт не нуждается в комментариях.
- **http://www.sommytech.com.ar** — сайт фирмы SommyTech, разрабатывающей технологию BalloonDialog.
- **http://subritto.h1.ru** — web-страница автора книги. Тут вы можете найти документацию по MS Agent и Speech API в более простом изложении. На web-странице существует гостевая книга, где можно задать любой вопрос автору, а также форум для обсуждения проблем и вопросов, связанных с технологиями MS Agent и Speech API.
- **http://www.stc.rus.net/rus** — центр речевых технологий.
- **http://www.native-speech.by.ru**, **http://www.native-speech.invitation.ru** — сайт "Внедрение речевых технологий" Николая Викторовича Бочарова.
- **http://art.bdk.com.ru/govor** — разнообразная информация по речевым технологиям, а также программам распознавания речи и синтеза речи.
- **http://www.ims.uni-stuttgart.de/~moehler/synthspeech/examples.html** — примеры синтеза речи (Examples of Synthesized Speech).

- **http://www.vector-ski.ru/vecs/govorilka** — web-страница программы Govorilka Антона Рязанова (Anton Ryazanov).
- **http://govorunplus.narod.ru** — web-страница программы "Говорун+", Алексея Владимировича Ермакова.
- **http://characters.narod.ru** — web-страница, посвященная технологии Microsoft Agent v2.0. Здесь вы найдете небольшую коллекцию персонажей, статьи, программы и средства разработки для этой технологии.

Ссылки на файлы

- Для работы только с анимированным персонажем без использования синтеза и распознавания речи с помощью MS Agent для систем Windows 98 нужно установить сам сервис Microsoft Agent (занимаемый объем памяти — 395 Кбайт):

  http://activex.microsoft.com/activex/controls/agent2/MS Agent.exe
- Библиотека русской локализации для MS Agent (128 Кбайт):

  http://activex.microsoft.com/activex/controls/agent2/AgtX0419.exe
- Чтобы работать с MS Agent, нужны персонажи, и скачиваются они отдельно. Вот ссылки на наиболее популярные из них (от Microsoft):

  Джин (1,6 Мбайт):

  http://download.microsoft.com/download/0/0/c/00cde5f8-321d-4325-baae-eb27f1bde85f/Genie.exe

  Маг (1,8 Мбайт):

  http://download.microsoft.com/download/1/d/b/1dbee406-9b5f-48c5-b901-dd1a3f3c4669/Merlin.exe

  Попугай (3,3 Мбайт):

  http://download.microsoft.com/download/a/f/5/af572f68-b83e-4e2c-8b0f-fd5fadf588e7/Peedy.exe

  Робот (2,2 Мбайт):

  http://download.microsoft.com/download/2/b/9/2b904bbb-c0b1-4840-b332-ba0615d1041e/Robby.exe
- Большая коллекция персонажей:

  http://desktopmates.com/gallery.html
- Редактор персонажей Microsoft Agent Character Editor (1 Мбайт):

  http://download.microsoft.com/download/d/f/a/dfa1b90c-f879-4b80-a31d-06924121d9c6/ace.exe

- Лингвистический редактор Microsoft Agent Linguistic Information Sound Editing Tool (6,2 Мбайт), сюда входит и пакет функций Speech API, и движок распознавания речи:

  http://download.microsoft.com/download/0/5/f/05f09d93-9cf7-4a5d-9f6c-15fa88c3d2a7/liset.exe

- Для работы с речевыми функциями, как в составе с MS Agent, так и для прямого обращения к Speech API, нужно установить этот сервис Microsoft SAPI 4.0a runtime binaries (824 Кбайт):

  http://activex.microsoft.com/activex/controls/sapi/spchapi.exe

- Дополнительно к пакету функций необходимо установить специальные модули (engine, движки), которые осуществляют синтез или распознавание речи. Нужно установить хотя бы один языковой TTS-модуль, чтобы использовать синтез речи.

  - Движки от Lernout&Hauspie (L&H):

    русский (2,9 Мбайт):

    http://activex.microsoft.com/activex/controls/agent2/lhttsrur.exe

    английский (2,5 Мбайт):

    http://activex.microsoft.com/activex/controls/agent2/lhttseng.exe

    французский (2,2 Мбайт):

    http://activex.microsoft.com/activex/controls/agent2/lhttsfrf.exe

    итальянский (2 Мбайт):

    http://activex.microsoft.com/activex/controls/agent2/lhttsiti.exe

    немецкий (2,6 Мбайт):

    http://activex.microsoft.com/activex/controls/agent2/lhttsdun.exe

  - Движки от фирмы Digalo (все они shareware-версии):

    русский (7,6 Мбайт):

    ftp://ftp.iif.hu/pub/mirror/ftp.winsite.com/pub/pc/win95/sounds/digalocorerus.exe

    английский (9,2 Мбайт):

    ftp://ftp.iif.hu/pub/mirror/ftp.winsite.com/pub/pc/win95/sounds/digalocoreeng.exe

- Движок распознавания речи Microsoft Speech Recognition Engine (6 Мбайт), поддерживает только контекстно-свободную грамматику:

  http://activex.microsoft.com/activex/controls/agent2/actcnc.exe

- Для удобного просмотра всех движков в системе можно установить апплет Панели управления Speech Control Panel (927 Кбайт):

  http://download.microsoft.com/download/c/9/e/c9ee5f5d-7631-4ee7-aee4-dbd22b2b1439/SpchCpl.exe

- Для того чтобы полностью разобраться со Speech API, можно скачать дистрибутив SpeechAPI 4 SDK (7,83 Мбайт). В нем находится полная (английская) документация по SAPI4, примеры программ на C++ и Basic, а также содержится пакет функций SAPI (тот, что составляет 824 Кбайт), движок распознавания речи Microsoft Speech Recognition Engine (без диктовки):

  http://download.microsoft.com/download/speechSDK/Install/4.0a/WIN98/EN-US/SAPI4SDK.exe

- Полная версия SDK — SpeechAPI 4 SDK Suite (40 Мбайт). Все то же самое, что и в предыдущем файле, но плюс к этому — набор англоязычных модулей синтеза речи для Microsoft (не те, что от фирмы L&H), движок распознавания речи от Microsoft с поддержкой диктовки, который "весит" около 20 Мбайт:

  http://download.microsoft.com/download/speechSDK/Install/4.0a/WIN98/EN-US/SAPI4SDKSUITE.exe

- Полная документация к Microsoft Agent SDK в формате CHM (708 Кбайт):

  http://download.microsoft.com/download/2/1/8/2183fc41-9313-4624-9e9e-25d26d0f7e19/docs.zip

- BalloonDialog 6.5 от SommyTech©. "Шаровоздушная" выноска для расширения функциональности обычного Balloon персонажа (2,5 Мбайт):

  http://www.sommytech.com.ar/downloads/balloondialog/BlnDialog.exe

- Версия 7.0 выноски BalloonDialog для платформы .NET Framework. Работает в Visual Studio 2002 (1,1 Мбайт):

  http://www.sommytech.com.ar/downloads/balloondialog7/BlnDialog7Setup.msi

- Версия 7.1 выноски BalloonDialog для Visual Studio 2003 (1,4 Мбайт):

  http://www.sommytech.com.ar/downloads/balloondialog71/BlnDialog71Setup.msi

- Для BalloonDialog v7.x существует программа для создания скинов (сменных обложек интерфейса) — BalloonDialog SkinMaker (315 Кбайт):

  http://www.sommytech.com.ar/downloads/bdskinmaker/bdskinmaker.zip

Пополнить набор ссылок можно, покопавшись в Интернете. HTML-файл со всеми ссылками имеется на прилагаемом компакт-диске.

Приложение 2

Описание компакт-диска

На диске размещены все рассмотренные в книге примеры, а также дополнительные модули. Описание структуры каталогов приведено в таблице ниже.

Многие проекты содержат достаточное количество комментариев. Все примеры создавались в Delphi 5, и затем те из них, которые отказывались работать без изменений в Delphi 6 и 7, соответствующим образом изменялись. Поэтому в некоторых каталогах проектов можно найти папки с проектами для разных версий Delphi, например, D5 и D6_D7.

Все примеры проверялись на компьютерах со следующими программными конфигурациями:

- Windows 98 SE, Internet Explorer v6.0, MS Office 2000, Delphi 5, VisualStudio 6, MS Agent v2.0 (Agent Server build 2.00.0.2202; Agent Control build 2.00.0.2115), Speech API 4.0.4.2512;
- Windows 2000 Profession SP3, Internet Explorer v6.0, MS Office 2000, Delphi 6, MS Agent 2.0 (Agent Server build 2.00.0.3422; Agent Control build 2.00.0.3422), Speech API 4.0.4.3405;
- Windows XP Home Edition, Internet Explorer v6.0, MS Office XP, Delphi 7, MS Agent 2.0 (Agent Server build 2.00.0.3422; Agent Control build 2.00.0.3422), Speech API 4.0.4.2512;

В Delphi 7 с "родными" модулями (файлы AgentObjects_TLB.pas и AgentServerObjects_TLB.pas) для MS Agent некоторые проекты отказывались корректно работать, однако при замене их на аналогичные модули от Delphi 5 они обретали полную функциональность.

Каталог	Описание
\Components	Компоненты для работы с MS Agent и Speech API
\MSAgent2	Microsoft Agent v2.0
\MSAgentInterDLLRus	Национальный модуль русского языка для MS Agent

(продолжение)

Каталог	Описание
\Components	
\Characters	Персонажи для MS Agent: Claude, Genie, Merlin, Milton, Peedy, Robby
\SpeechAPI	Пакет речевых функций Microsoft Speech API 4.0
\MicrosoftSpeechControl	Апплет Панели управления для Speech API (Microsoft Speech Control Panel 4.0)
\SpeechAPI_SDK4_Suite	Полная версия Speech API SDK 4.0 Suite со всеми движками, голосами, документацией и примерами программ от Microsoft
\TextToSpeechEngine	Движки синтеза речи от фирмы Lernout & Hauspie: английский, русский, итальянский, французский; от Elan: Набор англоязычных голосов
\SpeechRecognitionEngine	Движок распознавания английской речи Microsoft Speech Recognition Engine
\Exampels	Примеры проектов книги
\Chapter1	Примеры для главы 1
\Chapter2	Примеры для главы 2
\Chapter3	Примеры для главы 3
\Chapter4	Примеры для главы 4
\1_CreateCharacters	Примеры персонажей для разд. 4.2. Редактор персонажей Microsoft Agent Character Editor
\2_CreateLWV	Примеры лингвистических файлов к разд. 4.3. Лингвистический редактор Microsoft Linguistic Information Sound Editing Tool
\3_BalloonDialog	Примеры для разд. 4.4. Компонент BalloonDialog от SommyTech
\4_MSAgent&SpeechAPI	Пример для разд. 4.5. Совмещаем технологии MS Agent и Speech API
\TestChars_D5	Приложение для тестирования персонажей и лингвистических файлов к главе 4 для Delphi 5
\TestChars_D6-7	Приложение для тестирования персонажей и лингвистических файлов к главе 4 для Delphi 6 и Delphi 7
\html	Файлы HTML-оболочки компакт-диска

(окончание)

Каталог	**Описание**
\Includes	Заголовочные файлы MS Agent, Speech API и BalloonDialog
\MSAgent	В подкаталогах этой директории расположены файлы AgentObjects_TLB.pas и AgentServerObjects_TLB.pas, импортированные различными версиями Delphi (с 5 по 7).
	Внимание! Для Delphi 5 эти файлы переименованы без суффикса `_TLB`
\SpeechAPI	Заголовочные файлы для работы со Speech API
\forPAS	Модули Speech.pas и Spchtel.pas для Delphi
\forC++	Исходные модули Speech.h и Spchtel.h на C++
\Pas&TLB	Заголовочные файлы PAS и библиотеки типов TLB для обработки событий при работе с интерфейсом синтеза речи VoiceText API и интерфейсом распознавания речи VoiceCommand API в позднем связывании от фирмы Borland
\BalloonDialog	Заголовочные файлы BlnDialog_TLB.pas и BlnDialogT_TLB.pas для работы с зарегистрированной и Trial-версией компонента BalloonDialog. Модуль BlnConst.pas с необходимыми константами для BalloonDialog
\Programms	Дополнительные программы и приложения автора
\DemoAgent	Приложение, демонстрирующее пользовательский интерфейс MS Agent к главе 2
\HTTPServer	Простейший HTTP-сервер для тестирования проектов 15_WebAgent и 20_StopAllMeth из главы 2
\AgentReaderPro	Программа автора книги для чтения текста через различные модули синтеза речи, а также записи ее в аудиофайл с поддержкой технологии MS Agent и голосовых команд
\Animals1.7.2	Программа автора книги, представляющая собой самообучающуюся игру, которая угадывает задуманных вами животных. Использованы технологии MS Agent и синтеза речи

Список литературы

1. Борн Г. Реестр Windows® 98: Пер. с англ. — СПб.: БХВ-Петербург, 2001. — 496 с.
2. Дарахвелидзе П. Г., Марков Е. П., Котенок О. А. Программирование в Delphi 5. — СПб.: БХВ-Петербург, 2001. — 784 с.
3. Дарахвелидзе П. Г., Марков Е. П., Котенок О. А. Программирование в Delphi 7. — СПб.: БХВ-Петербург, 2003. — 784 с.
4. Елманова Н., Трепалин С., Тенцер А. Delphi и технология COM. — СПб.: Питер, 2003. — 697 с.
5. Колесниченко О. В., Шишигин И. В. Аппаратные средства PC. — СПб.: БХВ-Петербург, 2001. — 1024 с.

Предметный указатель

A

ACA 49, 124
ACD 357
Ace.exe 347
ACF 49, 124
ACS 49, 124
Activate 172, 178
Active 175
Active client 169
ActiveForm 39
ActiveX 38
AddRef 8, 17
Advanced Character Options 60
Agent Control 51, 67, 77, 85
Agent Server 51, 70
AgentPropertyChange 108
AnimationNames 67, 143
Audio Objects 233, 260, 317
AudioOutput 68
AudioPause 247
AudioResume 247
AutoGain 330
AutoPopupMenu 152, 153
AXCtrls 390

B

BalloonObject 67, 157
BalloonWidth 388
BorderFocus 393
Branching 362
ButtonPicture 390, 404
ButtonPressed 407
ButtonsCaptions 396

C

Callback-интерфейс 253
Character 67, 86, 109, 410
Character ID 74
Characters Object 67
    SRModeID 196
    TTSModeID 184
CharacterStudio 351
CLSID 10, 26, 37
CoCreateGUID 11
CoCreateInstance 10, 71
CoInitialize 84
COM 5
Command & Control 231
CommandObject 151
    Voice 200
    VoiceCaption 200
Commands (collection) 67, 147, 196
    Voice 200
    VoiceCaption 200
CommandsWindow 68
ComObj 242
Connected 79
Context-free grammar 316
Correction Window 311
CreateOleObject 34, 84
Custom Character 354

D

Default Character 354
Default Character Properties 94
DefaultCharacterChange 94
Dictation 231

Dictation grammar 316
Dictation Session 311
Digalo 226
DirectSound Audio Destination 261
DirectSound Audio Source 317
DirectSpeechRecognition API 316
DirectSpeechRecognition API 233
DirectSR 335
DirectTextToSpeech API 262
DirectTextToSpeech API 233, 235
Dragon Systems 220
DragonDictate 220

E

Early binding 32
Elan Informatique 215
EmptyParam 95
Engine 225
Engine Object 261
Enum 143
Exporting rule 334
ExtraData 348

F

Features 273
File Audio Destination 261, 294
File Audio Source 317
FindWindow 384
First Byte 215
Frames 345

G

GestureAt 118
Get 128
    Queue 131
    Type 129
GetCharacterEx 74
GetIdsOfNames 33
GetInterface 22
GetOleFont 390
GetOlePicture 390
GetWindowsDirectory 114
Glossary Entries 311
Grammar 316
Grammar Object 319
Grammar Compiler Object 318
GUID 8, 74
GUID_NULL 242, 269

H

HasOtherClients 169
HelpContextID 154
HelpFile 154
HelpModeOn 154
HideBalloonToMove 393

I

IAgent 73
IAgentCharacter 74
IAgentCharacterEx 74, 384
IAgentCtlAnimationNames 143
IAgentCtlAudioObject 181
IAgentCtlAudioObjectEx 180
IAgentCtlBalloon 157
IAgentCtlBalloonEx 157
IAgentCtlCharacterEx 86, 384
IAgentCtlCharacters 86
IAgentCtlCommand 151
IAgentCtlCommandEx 151
IAgentCtlCommands 147
IAgentCtlCommandsEx 147
IAgentCtlRequest 87
IAgentCtlUserInput 151, 201
IAgentEx 71
IAudio 261, 317
IAudioDestNotifySink 261
IaudioFile 261, 294
    DoSet 295
    Flush 296
    RealTimeSet 296
IAudioFileNotifySink 261, 296
IaudioMultiMediaDevice 261, 317
    DeviceNumSet 271
IAudioSource 317
IAudioSourceNotifySink 317
IconPicture 390
IDispatch 32

IDL 10
IdleComplete 117
IdleOn 117
IdleStart 117
Idling 116
IEnumVariant 144
IFontDisp 390
IListenCallback 308
Images 345
Importing list 334
Input-active 173, 177
Interfaces 273
International Phonetic Alphabet 374
Interrupt 134
IntroPhrase 348
Inverse Text Normalization 311
Invoke 33, 34
IPA 374
IPictureDisp 390
IRecordInfo 32
ISRAttributes 318, 330
ISRCentral 318
  GrammarLoad 319
ISRDialogs 318
ISREnum 317, 321, 409
ISRFind 317
ISRGramCFG 318
ISRGramCommon 318
ISRGramDictation 318
ISRGramInsertionGUI 318
ISRGramNotifySink 318, 321
  PhraseFinish 323
ISRNotifySink 318, 321
ISRSpeaker 312, 318
ISTGramComp 318
ITTSAttributes 261, 281, 409
  PitchGet 281
  SpeedGet 281
  VolumeGet 281
ITTSBufNotifySink 261, 289, 290
ITTSCentral 261, 269, 410
  Inject 288
  Register 293
  UnRegister 294
ITTSDialogs 261, 290
ITTSEnum 265, 408
  Find 276
  Next 266
  Select 268, 276
  Skip 266
ITTSNotifySink 261, 290
IUnknown 8, 17, 32
IVCmdAttributes 299, 304
IVCmdDialogs 299
IVCmdEnum 299
IVCmdMenu:
  Add 301
  MenuCreate 300
IVCmdNotifySink 303
  CommandRecognize 303
  VUMeter 303
IVDctAttributes 312
IVDctCommands 312
IVDctDialogs 312
IVDctGUI 312
IVDctInvTextNorm 312
IVDctNotifySink 312
  PhraseFinish 312
  PhraseHypothesis 312
  PhraseStart 312
IVDctText 312
IVoiceCallback 256
IVoiceCmd 299
IVoiceDictation 312
IVoiceText 236
IVTxtAttributes 236, 248
IVTxtAuto 255
IVTxtDialogs 240
IVTxtNotifySink 239

L

LanguageID 186, 411
Late binding 32
Learnout & Hauspie 218
Lernout & Hauspie 226
Lernout&Hauspie 185
Limited-domain grammar 316
Liset.exe 369
Listen 203
Listening Tip 203

Load 74, 79, 94
LWV 369

M

Microsoft Agent Character Editor 343
Microsoft Speech Recognition Engine 227
Microsoft Agent Control 2.0 52
Microsoft Agent v1.5 50
Microsoft Speech Control Panel 339
Microsoft Speech API 224
Microsoft Speech Recognition Engine 369
MMSystem модуль 263, 320
ModeID 266
ModeName 266
Monologue 215
MoveBalloonToCharacter 393
MoveCause 120
MoveStyle 393
MoveTo 119
MsgBalloon 383, 395
Msvcrt.dll 50
Multimedia Audio Destination 261
Multimedia Audio Source 317

N

Non-input-active 175
Notification sink 236
Notification sink 73

O

Office Assistant Character 354
OLE 5
OLE Automation 5, 31, 47
OleCheck 242
OLE-контейнер 31
OLE-объект 31
OLE-сервер 31
OnActivateInput 175
OnActiveClientChange 169
OnCommand 151
OnDeactivateInput 175
OnHide 120
OnListenComplete 204
OnListenStart 204
OnMove 120
OnRequestComplete 93
OnRequestStart 93
OnShow 120
OnVUMeter 309
OptionPicture 404
OptionPressed 407
OriginalHeight 110
OriginalWidth 110

P

Pitch 186
Play 117
PropertySheet 68, 107
ProVerbe 215
PTTSModeInfoRank 276
PWaveOutCaps 271

Q

QueryInterface 8, 22

R

RaiseRequestErrors 97
Regsvr32.exe 50
Release 8, 14, 17
ReminderPhrase 348
Request 68, 87
Request ID 74
ResetProperties 382
Return Animation 360
RGB 352

S

SDATA 268, 305
SetCharacterPosition 393
Sharing Object 234
Show 81
ShowDefaultCharacterProperties 94
SoundEffects 182
Speak 183, 192, 410

Speech Recognition Enumerator 317, 409
Speech.pas 230
SpeechInput 68
Speed 186
SREnumerator 317
SRModeID-свойство
(CharacterObject) 196
SRModeInfo 327
dwFeatures 329
dwInterfaces 330
dwSequencing 328
SRPHRASEA 324
SRStatus 196
Stop 137
StopAll 138
StopSpeaking 247

T

TAgent 70
Tags 284
TAutoObject 253
Telephony 231
Text to speech 231
TextData 410
Text-To-Speech 260
Text-To-Speech Enumerator 261, 408
TextToSpeech — элемент
управления 256
TFont 390
TGUID 11, 225
Think 121
ThunderRT6FormDC 384
Tip Window 59
Topics 311
TTSATTR_MINSPEED 281
TTSDATAFLAG_TAGGED 288
TTSFEATURE_TAGGED 284
TTSModeID 409
TTSModeID-свойство
(CharacterObject) 184
TTSModeInfo 266, 268, 273
dwFeatures 273
TTS-генераторы 225
TTTSModeInfoA 276
TVTSiteInfo 248

U

Unload 75, 81
UserMove 394

V

VCMDCOMMAND 301
Vdict.dll 315
ViaVoice 220
VisibilityCause 120
Voice Text API 234
Voice Command API 233
Voice Command API 233
Voice Dictation API 233
Voice Dictation API 233, 310
Voice Telephony API 234
Voice Telephony API 233
Voice Text API 233
VoiceCommand 298
VoiceCommandAuto.tlb 253
VoiceCommandAutoVar.tlb 307
VoiceCommandAutoVar_TLB.pas 307
VoiceMenu Enumerator 299
VoiceText 236
VoiceTextAuto_TLB.pas 253
VoiceTextAutoVar.tlb 253
VoiceTextAutoVar_TLB.pas 253
VText.dll 256

W

Wait 165
WAVE_MAPPER 264, 271
WaveInGetNumDevs 320
WaveOutGetDevCaps 271
DeviceID 271
WaveOutGetNumDevs 271
Wild-card 306
Windows Help Compiler 154

X

Xcommand.dll 309
Xlisten.dll 335

А

Активный клиент 169
Анимации возвратные 360
Анимация рта 366

Б

Библиотека:
типов 10
COM 9

Г

Глоссарий 311
Грамматика:
диктовки 316
контекстно свободная 316
ограниченного домена 316

Д

Движки 225

И

Импортируемые списки 334
Интерпретатор 218
Интерфейс 7

К

Компаратор 217
Контекстно свободная грамматика 316

Л

Лингвистические файлы 369
Лингвистический редактор 369

М

Модуль:
AXCtrls 390
ComObj 242
Speech.pas 230

Н

Наборы анимаций 345

О

Обратная нормализация текста 311
Окно:
About 225, 252
General 225, 252
Lexicon 225, 252
Training 225
TrainMic 225
Translate 225, 252
Оценка состояния запроса 88

П

Пакет речевых функций 180
Переходы 362
Персонаж:
по умолчанию 94
Input-active 173, 177
Non-input-active 175
Порог шума 331
Препроцессор 217

Р

Редактор персонажей 343

С

Сеансы диктовки 311

Сервер:
автоматизации 31
COM 9
Синтезаторы речи 225
Снимки 345
Состояния персонажа 116
Списки имен 305, 333

Т

Теги: 190, 284
LST 193
MAP 193
Топики 311

Ф

Фабрика класса 10
Файл персонажа 345
Фрейм 345

Э

Экспортируемые правила 334
Экстрактор 217
Элементы синтаксиса команд MS Agent 198
Эхо 331