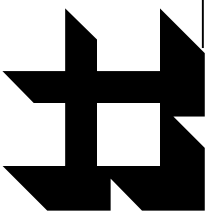

ГЛАВА 24

Мультимедиа

Возможно, у вас иногда возникало желание заставить ваши приложения "заговорить". К сожалению, встроенные возможности Visual FoxPro по воспроизведению звука и видео, надо сказать, никакие. Конечно, можно использовать различные ActiveX-компоненты; однако эта проблема достаточно просто решается средствами Windows API.

В этой главе вы узнаете, как записать и воспроизвести простые WAV-файлы, содержащие, например, ваши комментарии, научитесь работать с MCI (Multimedia Control Interface), позволяющим воспроизводить звуковые и видеофайлы различных форматов, а также познакомитесь с персонажем, имя которого может заинтриговать любого — мы имеем в виду Агента Microsoft.

Запись и воспроизведение WAV-файлов

В этом разделе мы рассмотрим самый простой способ создания звуковых файлов в стандартном рекордере Windows и их воспроизведение при помощи Windows API-функции PlaySound.

Запись звукового файла

Для записи коротких звуковых фрагментов вы можете воспользоваться стандартным рекордером Windows. Для его запуска нажмите кнопку **Пуск** на Рабочем столе Windows и в появившемся меню выберите пункты **Программы** | **Стандартные** | **Развлечения** | **Звукозапись**. Окно рекордера показано на рис. 24.1.

Вставьте в разъем звуковой карты микрофон и нажмите кнопку **Запись**. По окончании записи нажмите на кнопку **Стоп**, после чего в меню **Файл** рекордера выберите пункт **Сохранить как**. В появившемся диалоговом окне **Сохранение файла** (рис. 24.2) выберите папку и файл для сохранения записанного звукового фрагмента. По умолчанию этот фрагмент будет записан в формате *44,1 кГц, 16 бит, стерео*, что не совсем хорошо, т. к. такой файл будет занимать очень много места.

Выберите другой формат записи, нажав на кнопку **Изменить** диалогового окна **Сохранение файла**. Появится диалоговое окно **Выбор звука** (рис. 24.3). Выберите в нем более приемлемый формат, требующий гораздо меньше ресурсов, например, *8 кГц, 8 бит, моно*.

Таким образом, вы можете подготовить все необходимые звуковые фрагменты и потом использовать их в своих приложениях.

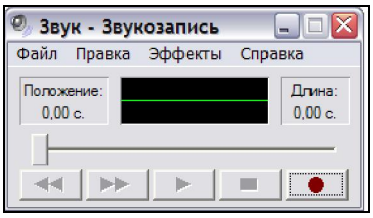

Рис. 24.1. Окно рекордера Windows

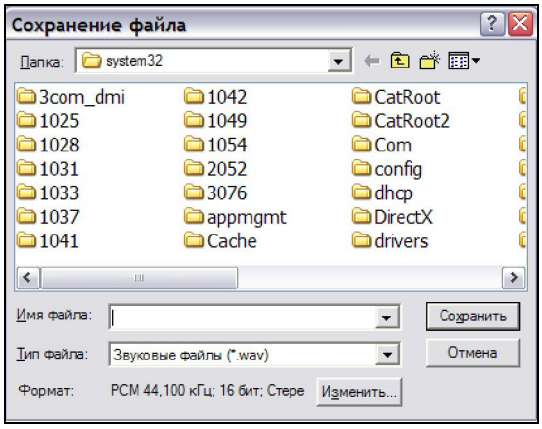

Рис. 24.2. Диалог **Сохранение файла**

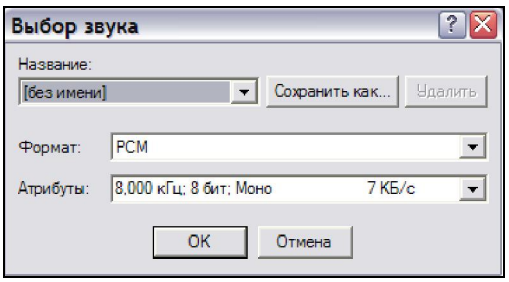

Рис. 24.3. Диалоговое окно **Выбор звука**

Воспроизведение звукового файла

Для воспроизведения звуковых файлов WAV-формата (а также звуков, оповещающих о событиях Windows) мы воспользуемся Windows API-функцией `PlaySound` из библиотеки Winmm.dll. Вот ее объявление в Visual FoxPro:

```
DECLARE Long PlaySound IN Winmm.dll ;
        String Sound, Long hmod, Long SoundType
```

Функция получает три параметра:

- *Sound* — имя WAV-файла или имя стандартного звука Windows;

- *hmod* — указатель на загруженный ресурс. При использовании функции в Visual FoxPro этот параметр должен быть равен нулю;
- *SoundType* — тип воспроизведения звука.

Значения параметра *SoundType* определяется как сумма значений, перечисленных в табл. 24.1.

Таблица 24.1. *Значения параметра SoundType функции PlaySound*

Имя константы в MSDN	Значение	Описание
SND_SYNC	0x00000000	Файл воспроизводится синхронно; функция не возвращает управление программе до окончания воспроизведения
SND_ASYNC	0x00000001	Файл воспроизводится асинхронно и функция возвращает управление сразу после начала воспроизведения. Для того чтобы прервать воспроизведение, нужно вызвать функцию с пустым именем файла
SND_LOOP	0x00000008	Используется только совместно с флагом SND_ASYNC. Файл будет воспроизводиться бесконечное количество раз. Для того чтобы прервать воспроизведение, нужно вызвать функцию с пустым именем файла
SND_NOSTOP	0x00000010	Перед началом воспроизведения функция проверяет, не воспроизводится ли какой-либо файл в момент ее вызова. Если ресурс, необходимый для генерации звука, занят, функция возвращает ноль. Если этот флажок не определен, то функция пытается прекратить проигрывание воспроизводимого в данный момент звукового фрагмента и начать воспроизведение нового звукового фрагмента
SND_ALIAS	0x00010000	Параметр *Sound* содержит наименование звука события системы
SND_FILENAME	0x00020000	Параметр *Sound* содержит имя WAV-файла

Функция возвращает ноль при успешном завершении и значение, отличное от нуля, в случае ошибки.

В следующем фрагменте кода показано, как при помощи этой функции воспроизвести звуковой файл.

```
#DEFINE SND_FILENAME 0x00020000
DECLARE Long PlaySound IN Winmm.dll String, Long, Long
cWavFile = GETFILE('WAV')
IF !EMPTY(cWavFile)
   nResult = PlaySound(cWavWile, 0, SND_FILENAME)
ENDIF
```

Следующий фрагмент кода демонстрирует бесконечно повторяющееся воспроизведение звукового файла:

```
#DEFINE SND_ASYNC        0x00000001
#DEFINE SND_LOOP 0x00000008
#DEFINE SND_FILENAME     0x00020000
```

```
DECLARE Long PlaySound IN Winmm.dll String, Long, Long
cWavFile = GETFILE('WAV')
IF !EMPTY(cWavFile)
   nResult = PlaySound(cWavWile, 0, SND_ASYNC + SND_LOOP + SND_FILENAME)
ENDIF
```

Остановить воспроизведение можно, вызвав функцию `PlaySound` с пустым значением параметра *Sound*:

```
PlaySound(NULL, 0, 0)
```

Воспроизведение стандартных звуков Windows

Windows включает в себя набор стандартных звуков, которые воспроизводятся при возникновении различных событий, например, при регистрации пользователя или очистке Корзины. В табл. 24.2 перечислены наименования некоторых из этих звуков.

Таблица 24.2. *Наименования некоторых стандартных звуков Windows*

Chimes	Ding	Recycle	SystemAsterisk	SystemHand
Chord	MailBeep	Ringin	SystemExclamation	SystemStart
DeviceFail	Notify	Ringout	SystemExit	Tada

В следующем фрагменте кода демонстрируется воспроизведение звука, который обычно воспроизводится после регистрации пользователя в системе:

```
#DEFINE SND_ALIAS 0x00010000
DECLARE Long PlaySound IN Winmm.dll String, Long, Long
nResult = PlaySound('SystemStart', 0, SND_ALIAS)
```

Звуковой WAV-файл, передаваемый функции для воспроизведения, перед проигрыванием полностью загружается в оперативную память компьютера. Поэтому желательно, чтобы такой файл имел не очень большие размеры. Для воспроизведения аудио- и видеофайлов большого размера (до сотен мегабайт) можно воспользоваться средствами интерфейса управления мультимедиа — MCI.

Интерфейс управления мультимедиа (MCI)

В Windows имеется набор компонентов, объединенных одним общим названием — MCI, или *Multimedia Control Interface*, что можно перевести как *"Интерфейс управления устройствами мультимедиа"*. Назначение этого интерфейса — управление записью и воспроизведением аудио- и видеофайлов.

MCI при необходимости использует различные кодеки (которые, естественно, должны быть установлены) для кодирования и декодирования "сжатых" файлов, таких как MP3, WMA или AVI, что делает ее возможности практически неограниченными.

Управление мультимедийными устройствами и файлами в MCI отличается чрезвычайной простотой: вам потребуется всего лишь две Windows API-функции, первая из которых, `mciSendString`, позволяет управлять записью и воспроизведением звука и видео, а вторая, `mciGetErrorString`, возвращает описание ошибки MCI по ее коду.

Объявление функции `mciSendString`:

```
DECLARE Long mciSendString IN WinMM.DLL String MCIString, ;
        String @ RetMsg, Long RetMsgSize, Long hInstance
```

Параметры функции:

- *MCIString* — строка, которая содержит команду для MCI;
- *RetMsg* — строка, в которую помещается результат выполнения команды. Обычно длина этой строки составляет несколько десятков символов;
- *RetMsgSize* — длина строки *RetMsg*;
- *hInstance* — дескриптор окна повторного вызова. В Visual FoxPro не поддерживается, поэтому параметр должен быть равен нулю.

Функция возвращает нуль при успешном завершении или код ошибки в случае неудачи.

Объявление функция `mciGetErrorString`:

```
DECLARE Long mciGetErrorString IN WinMM.DLL ;
        Long ErrorCode, String Buffer, Long BufferSize
```

Параметры функции:

- *ErrorCode* — код ошибки, возвращенный функцией `mciSendString`;
- *Buffer* — строка, в которую помещается текст сообщения об ошибке;
- *BufferSize* — размер строки *Buffer*.

Строка, передаваемая MCI функцией `mciSendString`, имеет следующий формат:

- имя команды;
- алиас файла или устройства;
- имя параметра (необязательно);
- значение параметра (необязательно).

Все элементы строки разделяются пробелами.

> ***Замечание***
>
> *Алиас* — это внутреннее имя, которое вы присваиваете конкретному файлу или устройству при его открытии в команде `OPEN`. Это имя затем используется всеми командами MCI для управления процессом записи/воспроизведения.

Команды MCI

В Windows XP интерфейс MCI поддерживает 46 команд. Конечно, вы вряд ли будете когда-либо использовать их все; вы же не ставите себе задачу создать на Visual FoxPro звуковой или видеопроцессор. Но вот возможность демонстрировать в окне формы фрагменты видео или реагировать на действия пользователя не только ограниченным набором стандартных звуков Windows, вероятно, будет весьма полезной.

Мы рассмотрим здесь следующие команды MCI (табл. 24.3).

Таблица 24.3. *Некоторые команды MC*

Команда	Выполняемое действие
OPEN	Открывает файл
PLAY	Запускает процесс воспроизведения
PAUSE	Приостанавливает процесс воспроизведения
CLOSE	Закрывает файл
STATUS	Позволяет получить информацию о размере файла, текущей позиции воспроизведения и т. д.
SEEK	Выполняет переход в начало, конец или в заданную позицию файла
SETAUDIO	Выполняет установки для звука (громкость и т. д.)
SETVIDEO	Выполняет установки для видео (яркость, контрастность и т. д.)
WHERE	Используется только при воспроизведении видео. Позволяет определить размеры кадра изображения (в пикселах)

Команда *OPEN*

Команда открывает файл и присваивает ему алиас. При открытии видеофайла в команде нужно указать, где он должен воспроизводиться — непосредственно на экране монитора, в собственном окне или в окне формы.

Формат команды для открытия аудиофайла:

```
OPEN "имя_файла" ALIAS алиас WAIT
```

Формат команды для открытия видеофайла

```
OPEN "имя_файла" ALIAS алиас STYLE значение WAIT
```

Значения параметра `STYLE` перечислены в табл. 24.4.

Таблица 24.4. *Значения параметра* `STYLE` *команды* `OPEN`

Параметр	Положение области вывода
Popup	Видео выводится непосредственно на экран монитора (этот режим удобен для полноэкранного просмотра фильмов, но применение его в приложениях на Visual FoxPro вряд ли оправдано)
Overlapped	Видео выводится в отдельном окне, которое можно перемещать и размеры которого можно менять
child parent HWND	Видео выводится в окне формы с дескриптором HWND

Замечание

Так как команда передается MCI как строка символов, то значение HWND формы так же должно быть преобразовано к символьному типу.

В следующем фрагменте кода показано, как открыть файл *d:\Sounds\MySong.mp3*:

```
cCmd = 'OPEN "d:\Sounds\MySong.mp3" ALIAS Song WAIT'
cRetMsg = space(80)
nStatus = mciSendString(cCmd, cRetMsg, LEN(cRetMsg))
```

А следующий фрагмент кода показывает, как открыть видеофайл, который будет выводиться в собственном окне:

```
cCmd = 'OPEN "d:\Films\MyVideo.avi" ALIAS Video Style overlapped WAIT'
cRetMsg = space(80)
nStatus = mciSendString(cCmd, cRetMsg, LEN(cRetMsg))
```

Обратите внимание, что в первом примере файлу присваивается алиас Song, а во втором — Video. Естественно, вы можете использовать любые другие имена для обозначения алиаса.

Замечание

Вы можете указывать параметр STYLE и при открытии звукового файла; MCI просто проигнорирует его.

Команда *PLAY*

Запускает процесс воспроизведения файла. Формат команды:

```
PLAY алиас
```

Команда *PAUSE*

Команда приостанавливает воспроизведение. Формат команды:

```
PAUSE алиас
```

Команда *CLOSE*

Воспроизведение звука и видео в MCI выполняется в асинхронном режиме, причем процесс воспроизведения никоим образом не связан с вашим приложением. Поэтому

вполне возможна ситуация, когда пользователь завершил приложение, а процесс воспроизведения звука продолжается. Завершить воспроизведение можно, только послав MCI команду CLOSE. Эта команда имеет следующий формат:

```
CLOSE алиас WAIT
```

Команда *STATUS*

Эта команда позволяет получить информацию о текущем состоянии процесса воспроизведения. Формат команды:

```
STATUS алиас параметр
```

где *параметр* — наименование параметра, значение которого будет возвращено функцией mciSendString в ее параметре *RetMsg*.

Из всех возможных параметров, значения которых можно получить от MCI, рассмотрим только те, которые связаны с воспроизведением звука и видео. Эти параметры перечислены в табл. 24.5.

Таблица 24.5. *Параметры, значения которых можно получить командой STATUS*

Наименование параметра	Выполняемое действие и возвращаемые значения
Mode	"playing" — выполняется воспроизведение "paused" — пауза "stopped" — файл загружен, воспроизведение еще не начато
length	Возвращает максимальное количество единиц, определяющих продолжительность аудио- или видеофайла
Position	Возвращает текущую позицию в единицах, определяющих продолжительность аудио- или видеофайла
Ready	Возвращает "true", если устройство готово к приему следующей команды
window handle wait	Возвращает hWnd окна, в котором отображается видео

В следующем фрагменте кода показано, как получить значение текущей позиции при воспроизведении файла с алиасом *Song*:

```
cRetMsg = SPACE(40)
mciSendString('STATUS Song position', @cRetMsg, 40, 0)
? VAL(cRetMsg)
```

Команда *SEEK*

Команда выполняет позиционирование внутри файла. Формат команды:

```
SEEK алиас параметр [значение]
```

Некоторые из возможных параметров команды приведены в табл. 24.6.

Таблица 24.6. *Параметры команды* `SEEK`

Наименование параметра	Выполняемое действие
`to start`	Переход к началу файла
`to end`	Переход в конец файла
`to` *значение*	Переход на заданную позицию. Значение позиции должно находиться в интервале от 0 до значения, возвращенного командой `STATUS length`

Следующий фрагмент кода демонстрирует использование команды `SEEK` для перехода в нужную позицию файла с алиасом *Song*:

```
cCMD = 'SEEK Song to ' + LTRIM(STR(nPosition))
cRetMsg = SPACE(40)
mciSendString(cCMD, @cRetMsg, 40, 0)
```

Команда *SETAUDIO*

Команда управляет воспроизведением звука. Формат команды:

```
SETAUDIO алиас параметр [значение]
```

Параметры команды приведены в табл. 24.7.

Таблица 24.7. *Параметры команды* `SETAUDIO`

Наименование параметра	Выполняемое действие
`on, off`	Включает / выключает звук
`left on, left off`	Включает / выключает звук левого канала
`right on, right off`	Включает / выключает звук правого канала
`volume to` *значение*	Устанавливает уровень звука. Значение параметра должно находиться в интервале от 0 до 1000
`left volume to` *значение*	Устанавливает уровень звука в левом канале
`right volume to` *значение*	Устанавливает уровень звука в правом канале

В следующем фрагменте кода показано, как установить уровень звука при воспроизведении равным 20 децибелам:

```
cCMD = 'SETAUDIO Song volume to 100'  && уровень 20 децибел
cRetMsg = SPACE(40)
mciSendString(cCMD, @cRetMsg, 40, 0)
```

Команда *SETVIDEO*

Команда управляет некоторыми параметрами при воспроизведении видео. Вот ее формат:

```
SETVIDEO алиас параметр [значение]
```

Параметры команды приведены в табл. 24.8.

Таблица 24.8. *Параметры команды* `SETAUDIO`

Наименование параметра	Выполняемое действие
`Algorithm` *значение*	Определяет алгоритм сжатия для воспроизводимого изображения. MCI автоматически распознает алгоритмы *mpeg* и *h261*. Если для файла использовался другой алгоритм сжатия, то вы можете указать его здесь (для правильного воспроизведения видео в системе должен быть установлен соответствующий кодек)
`color to` *значение*	Устанавливает яркость изображения

Таблица 24.8 *(окончание)*

Наименование параметра	Выполняемое действие
`contrast to` *значение*	Устанавливает контрастность изображения
`gamma to` *значение*	Устанавливает насыщенность изображения
`on, off`	Разрешает / запрещает вывод видео

Команда *WHERE*

Из всех параметров команды интерес для нас представляет параметр `destination`, который возвращает размеры загруженного изображения. Размеры изображения возвращаются как четыре разделенные пробелом целых числа, определяющих координаты левого верхнего угла, ширину и высоту прямоугольной области видеокадра.

В следующем фрагменте кода показано, как можно определить размеры видеокадра (предполагается, что воспроизводимому файлу присвоен алиас *Video*).

```
cRetMsg = SPACE(40)
mciSendString('WHERE Video destination', @cRetMsg, 40, 0)
nWidth = VAL(GETWORDNUM(cRetMsg, 3, " "))    && Ширина и
nHeight = VAL(GETWORDNUM(cRetMsg, 4, " "))   && высота изображения
```

Особенности вывода видео на форме

Если вы хотите выводить видео на форму (а не на экран или в собственное окно), то вам, как минимум, нужно указать положение и размеры области вывода. Если вы не сделаете этого, то изображение будет отображаться в левом верхнем углу формы.

Определить положение области для вывода видео можно при помощи Windows API-функции SetWindowPos, которая, получив дескриптор окна, в котором это видео отображается, устанавливает его координаты и размеры. Вот объявление этой функции:

```
DECLARE Long SetWindowPos IN User32.dll ;
        Long hWnd, Long WndInsertAfter, Long left, Long top, ;
        Long width, Long height, Long flag
```

Передаваемые функции параметры:

- *hWnd* — дескриптор окна, в котором выводится видео;
- *WndInsertAfter* — определяет место окна вывода видео в Z-последовательности клиентских окон. В нашем случае оно должно располагаться над самым верхним окном формы (HWND_TOP), т. е. параметр должен быть равен нулю;
- *left*, *top* — координаты левой верхней точки области в пикселах;
- *width*, *height* — ширина и высота области;
- *flag* — параметр, определяющий возможности модификации окна.

Эта функция при успешном завершении возвращает отличное от нуля значение и нуль в случае ошибки.

Получить дескриптор окна, в котором выводится видео, можно, послав MCI команду STATUS с параметром *window handle wait*.

В листинге 24.1 показан код процедуры, которая выводит видео в заданной области формы. Имя видеофайла, hWND формы и размеры области для вывода видео передаются процедуре как параметры.

```
LPARAMETERS tcFile, tHWND, tnLeft, tnTop, tnWidth, tnHeight
LOCAL lcCMD, lcRetMsg, lnStatus, lnWidth, lnHeight, lnDestHwnd
DECLARE Long mciSendString IN WinMM.DLL String, String @, Long, Long
DECLARE Long SetWindowPos IN User32.dll ;
        Long, Long, Long, Long, Long, Long, Long
DECLARE Long mciGetErrorString IN WinMM.DLL Long, String, Long
* Открыть видеофайл для воспроизведения
lcCMD = 'OPEN "' + tcFile + '" ALIAS Video STYLE child parent ' + ;
        LTRIM(STR(tHWND))
lcRetMsg = SPACE(40)
lnStatus = mciSendString(lcCMD, @lcRetMsg, 40, 0)
IF lnStatus = 0
* Получить HWND окна, в котором выводится видео
   lcRetMsg = SPACE(40)
   lnStatus = mciSendString('STATUS Video window handle wait', ;
                            @lcRetMsg, 40, 0)
ENDIF
IF lnStatus = 0
* Изменить положение и размеры окна, в котором выводится видео
   lnDestHwnd = VAL(lcRetMsg)
```

```
   = SetWindowPos(lnDestHwnd, 0, tnLeft, tnTop, tnWidth, tnHeight, 0)
* Начать воспроизведение
   lcRetMsg = SPACE(40)
   lnStatus = mciSendString('PLAY Video', @lcRetMsg, 40, 0)
ENDIF
IF lnStatus != 0
* Если имела место ошибка, то сообщаем о ней
   lcRetMsg = SPACE(250)
   = mciGetErrorString(lnStatus, @lcRetMsg, 250)
   = MESSAGEBOX(ALLTRIM(lcRetMsg))
ENDIF
```

Класс для управления MCI

В папке Ffc, расположенной в корневой папке Visual FoxPro, вы найдете библиотеку классов _multimedia.vcx. Она содержит два класса, один из них предназначен для воспроизведения звука, а второй — для воспроизведения видео. Несмотря на очевидную простоту работы с MCI, эти классы достаточно сложны для понимания (и, как обычно, практически не документированы), поэтому в этом разделе мы создадим собственный класс, предназначенный для воспроизведения звука и видео в ваших приложениях.

Создайте библиотеку классов с именем `vfpmci`, добавьте нее класс `vfpmci`, в качестве класса-родителя выберите базовый класс `Custom`.

Добавьте в класс защищенное свойство `Status`, которое будет использоваться для хранения кода выполнения функции `mciSendString`. Установите его начальное значение равным нулю.

Добавьте еще одно свойство с именем `lLoad`, которое будет содержать логическое значение, определяющее, загружен ли в настоящий момент какой-либо аудио- или видеофайл или нет.

Метод *Init*

В методе `Init` класса объявите все необходимые Windows API-функции (листинг 24.2).

```
DECLARE Long mciSendString IN WinMM.DLL String, String @, Long, Long
DECLARE Long SetWindowPos IN User32.dll ;
        Long, Long, Long, Long, Long, Long, Long
DECLARE Long mciGetErrorString IN WinMM.DLL Long, String, Long
```

Метод *DoCMD*

Добавьте в класс защищенный метод `DoCMD` и введите в него код из листинга 24.3. Этот метод получает строку, которую необходимо передать MCI. Он вызывает функ-

цию `mciSendString` и обрабатывает возвращаемый ею код. В случае если команда выполнена успешно, функция возвращает строку, содержащую результат (значение параметра *`RetMsd`* — см. объявление функции). В противном случае выводится диалоговое окно **MESSAGEBOX** с сообщением об ошибке и метод возвращает пустую строку.

```
LPARAMETERS tcMCICommand
LOCAL lcRetMsg
lcRetMsg = SPACE(40)
this.Status = mciSendString(tcMCICommand, @lcRetMsg, 40, 0)
IF this.Status != 0
   lcRetMsg = SPACE(250)
   = mciGetErrorString(this.Status, @lcRetMsg, 250)
   = MESSAGEBOX(ALLTRIM(lcRetMsg))
   RETURN ""
ENDIF
RETURN alltrim(lcRetMsg)
```

Метод *Open*

Метод `Open` открывает переданный ему файл. Он получает следующие параметры:

- имя аудио- или видеофайла;
- `HWND` формы, в которой должно отображаться видео. Если параметр опущен, то видео будет выводиться в отдельном окне.

Добавьте метод в класс. Его код приведен в листинге 24.4.

```
LPARAMETERS tcFileName, tHWND
LOCAL lcCMD
IF VARTYPE(tcFileName) = "C" .and. FILE(tcFileName)
* Проверить, загружен ли в настоящий момент
* аудио- или видеофайл; если да, то закрыть его.
   IF this.lLoad
      this.DoCMD("CLOSE mediafile")
   ENDIF
   tHWND = IIF(VARTYPE(tHWND) = "N", tHWND, 0)
* Если значение HWND формы передано методу, то видео будет выводиться
* в окне формы; иначе — в отдельном окне. При воспроизведении аудиофайла
* MCI игнорирует параметр STYLE
   lcCMD = 'OPEN "' + tcFileName + '" ALIAS mediafile STYLE '
   IF tHWND = 0
      lcCMD = lcCMD + "Overlapped"
   ELSE
      lcCMD = lcCMD + "child parent " + LTRIM(STR(tHWND))
   ENDIF
   lcCMD = lcCMD + " WAIT"
   this.DoCMD(lcCMD)
```

```
   IF this.Status = 0
      this.lLoad = .t.
   ENDIF
ENDIF
```

Метод присваивает воспроизводимому файлу алиас *mediafile*.

В следующем фрагменте кода создается объект — экземпляр класса `vfpmci` и открывается для воспроизведения файл mysong.mp3:

```
oMCI = CREATEOBJECT("vfpmci")
oMCI.Open("d:\Моя любимая музыка\mysong.mp3")
```

А следующий фрагмент кода демонстрирует, как открыть видеофайл из метода `Click` командной кнопки, расположенной на форме. Предполагается, что видео будет воспроизводиться в окне этой формы:

```
oMCI.Open("d:\Мои видео\myfilm.avi", thisform.hWnd)
```

Методы *Play*, *Pause* и *Stop*

Так как метод `DoCMD` объявлен в нашем классе как защищенный, то нам необходимо добавить в класс три метода, управляющие воспроизведением файла: `Play`, `Pause` и `Stop`. Выполняемые ими действия понятны из их названий (листинги 24.5—24.7).

```
IF this.lLoad
   this.DoCMD("PLAY mediafile")
ENDIF
```

```
IF this.lLoad
   IF this.DoCMD("STATUS mediafile MODE") = "playing"
      this.DoCMD("PAUSE mediafile")
   ENDIF
ENDIF
```

```
IF this.lLoad
   this.DoCMD("CLOSE mediafile")
   this.lLoad = .f.
ENDIF
```

Метод *GetSize*

Метод возвращает размер воспроизводимого файла в единицах, используемых в текущем формате мультимедиа. Это могут быть миллисекунды (для аудио) или фреймы (для видео). Предполагается, что вас не интересует время воспроизведения файла, а

нужно лишь отслеживать его текущую позицию — тогда не имеет значения, в каких единицах выражен размер. В противном случае вам следует обратиться к документации по MCI в MSDN для изучения команд, позволяющих использовать форматы времени.

Добавьте метод в класс. Его код приведен в листинге 24.8.

```
LOCAL lcRetMsg, lnSize
lnSize = 0
IF this.lLoad
   lcRetMsg = this.DoCMD("STATUS mediafile LENGTH")
   IF this.Status = 0
      lnSize = VAL(lcRetMsg)
   ENDIF
ENDIF
RETURN lnSize
```

Вызов метода демонстрируется в следующем фрагменте кода:

```
nSize = oMedia.GetSize()
```

Метод *GetPosition*

Метод возвращает значение текущей позиции в воспроизводимом файле.

Добавьте метод в класс. Его код приведен листинге 24.9.

```
LOCAL lcRetMsg, lnPosition
lnPosition = 0
IF this.lLoad
   lcRetMsg = this.DoCMD("STATUS mediafile POSITION")
   IF this.Status = 0
      lnPosition = VAL(lcRetMsg)
   ENDIF
ENDIF
RETURN lnPosition
```

Вызов метода демонстрируется в следующем фрагменте кода:

```
nPosition = oMedia.GetPosition()
```

Метод *SetPosition*

Метод выполняет позиционирование в воспроизводимом файле. Он получает один параметр, который может принимать следующие значения:

- –1 — файл перематывается к началу;

- –2 — файл перематывается к концу;
- > 0 — любое положительное значение определяет позицию в файле.

Добавьте метод в класс. Его код показан в листинге 24.10.

```
LPARAMETERS tnPosition
LOCAL lcCMD, lnSize
IF VARTYPE(tnPosition) = "N"
   IF this.lLoad
      lcCMD = "SEEK mediafile to "
      DO CASE
         CASE tnPosition = -1    && В начало файла
            lcCMD = lcCMD + "start"
         CASE tnPosition = -2    && В конец файла
            lcCMD = lcCMD + "end"
         CASE tnPosition >= 0    && На заданную позицию
            IF tnPosition <= this.GetSize()
               lcCMD = lcCMD + LTRIM(STR(tnPosition))
            ELSE
               lcCMD = lcCMD + LTRIM(STR(this.GetSize()))
            ENDIF
      ENDCASE
      this.DoCMD(lcCMD)
   ENDIF
ENDIF
```

При установке на указанную позицию проверяется допустимость значения переданного методу параметра. Его значение не должно быть меньше нуля и больше максимального размера файла.

Метод *SetVolume*

Метод выполняет установку уровня звука одновременно во всех каналах. Если вы хотите регулировать звук в каждом канале отдельно или устанавливать баланс между каналами, то используйте соответствующие команды, перечисленные в табл. 24.7.

Добавьте метод в класс. Он получает один параметр, устанавливающий уровень воспроизводимого звука. Код метода приведен в листинге 24.11.

```
LPARAMETERS tnVolume
IF VARTYPE(tnVolume) = "N"
   IF tnVolume < 0
      tnVolume = 0
   ENDIF
   IF tnVolume > 1000
      tnVolume = 1000
```

```
   ENDIF
   IF this.lLoad
      this.DoCMD("SETAUDIO mediafile volume to " + LTRIM(STR(tnVolume)))
   ENDIF
ENDIF
```

Следующий фрагмент кода показывает, как установить уровень воспроизведения, равный 75% от максимально возможного:

```
oMedia.SetVolume(750)
```

Метод *GetVideoDimension*

Метод определяет и возвращает размеры кадра видеоизображения. Он получает два передаваемых по ссылке параметра и записывает в них значения ширины и высоты кадра в пикселах.

Добавьте метод в класс. Его код приведен в листинге 24.12. Вызов метода при воспроизведении звукового файла приводит к ошибке.

```

PARAMETERS tnWidth, tnHeight
LOCAL lcRetMsg
IF this.lLoad
   lcRetMsg = this.DoCMD('WHERE mediafile destination')
   IF this.Status = 0
      tnWidth = VAL(GETWORDNUM(lcRetMsg, 3, " "))
      tnHeight = VAL(GETWORDNUM(lcRetMsg, 4, " "))
   ENDIF
ENDIF
```

В следующем фрагменте кода показан пример использования метода.

```
nWidth = 0
nHeight = 0
oMedia.GetVideoDimension(@nWidth, @nHeight)
```

Метод *SetVideoRect*

Метод устанавливает размеры прямоугольной области для вывода видео. Он получает четыре параметра: координаты левой верхней точки, ширину и высоту области.

Добавьте метод в класс. Его код приведен в листинге 24.13.

```

LPARAMETERS tnLeft, tnTop, tnWidth, tnHeight
IF VARTYPE(tnLeft) + VARTYPE(tnTop) + VARTYPE(tnWidth) + ;
   VARTYPE(tnHeight) = "NNNN"
* Получить HWND для окна, в котором выводится видео
```

```
   lcRetMsg = SPACE(40)
   lnStatus = mciSendString('STATUS Video window handle wait', ;
                            @lcRetMsg, 40, 0)
   IF lnStatus = 0
* Изменить положение и размеры окна, в котором выводится видео
      lnDestHwnd = VAL(lcRetMsg)
      IF SetWindowPos(lnDestHwnd, 0, tnLeft, tnTop, ;
                      tnWidth, tnHeight, 0) = 0
         = MESSAGEBOX("Ошибка при изменении размеров окна для видео", ;
                      64, "MCI")
      ENDIF
   ENDIF
ENDIF
```

Если вы хотите отображать видео в заданной области на форме, то должны вызвать этот метод после вызова методов `Open` и `GetVideoDimension`. Вы должны выполнить необходимые вычисления, если размеры области вывода видео будут отличаться от размеров видеокадра, для того, чтобы не допустить геометрических искажений.

Метод *Destroy*

Метод `Destroy` должен содержать код, прекращающий воспроизведение. В этом коде должен вызываться метод `Stop` класса:

```
this.Stop()
```

Проигрыватель звуковых файлов

Не подумайте, что мы решили заняться изобретением очередного велосипеда. Нет, наша цель гораздо проще: на примере простого мультимедийного проигрывателя показать, как можно применить в ваших приложениях описанный выше класс `vfpmci`.

Добавьте в библиотеку классов `vfpmci` новый класс с именем `vfpplayer`; в качестве класса-родителя выберите `Container`. Таким образом, мы создадим контейнерный класс, который будет содержать все необходимые компоненты для управления воспроизведением аудио- и видеофайлов.

Разместите в контейнере четыре командные кнопки и установите значения их свойств *`Name`* и *`Caption`* так, как показано в табл. 24.9.

Таблица 24.9. *Командные кнопки класса `vfpplayer`*

Значение свойства `Name`	**Значение свойства `Caption`**	**Назначение кнопки**
cmdOpen	Открыть	Открывает файл
cmdPlay	Играть	Начинает воспроизведение
cmdPause	Пауза	Приостанавливает воспроизведение

cmdStop	Стоп	Останавливает воспроизведение, закрывает файл

Разместите в контейнере еще два компонента — ActiveX-управляющие элементы **Microsoft Slider Control** (слайдеры), лучше пятой версии, потому что они поддерживают темы Windows XP. Первый из них будет выполнять роль управляемой шкалы, индицирующей ход процесса воспроизведения, а второй позволит регулировать громкость звука. Дайте этим компонентам имена **Indicator** и **Volume**. Для компонента с именем **Volume** необходимо установить значение свойства `Max` равным 1000 (максимально допустимый уровень громкости), а значение свойства `Value` равным 700 (70% от максимального уровня). Присвойте свойствам `TickStyle` объектов `Slider Control` значение 3 (No Ticks).

И, наконец, разместите в контейнере таймер; установите его свойство `Enabled` в "ложь", а свойству `Interval` присвойте значение 500 (таймер будет срабатывать каждые полсекунды). Таймер поможет нам отслеживать процесс воспроизведения.

Скомпонуйте компоненты контейнера так, чтобы получилось примерно то, что показано на рис. 24.4.

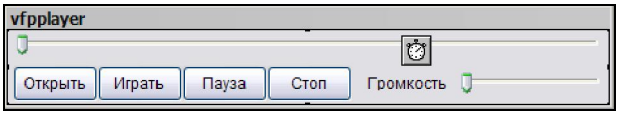

Рис. 24.4. Вид объекта `vfpplayer` в Конструкторе классов

Добавьте в контейнер свойство `oMedia`. Это свойство будет использоваться для хранения ссылки на объект — экземпляр класса `vfpmci`.

В метод `Init` контейнера введите код из листинга 24.14.

```
this.oMedia = CREATEOBJECT("vfpmci")
IF VARTYPE(this.oMedia) != "O"
   RETURN .f.
ENDIF
```

В метод `Click` кнопки **Открыть** введите код из листинга 24.15.

```
LOCAL lcFile
lcFile = GETFILE("MP3|WMA|WAV")
IF !EMPTY(lcFile)
   WITH this.Parent
      .oMedia.Open(lcFile)
      .oMedia.SetVolume(750)            && Начальный уровень громкости
```

```
      .Interval.value = 0               && Установка начального и
      .Interval.max = .oMedia.GetSize() && максимального значения шкалы
      .Timer1.Enabled = .t.             && Включить таймер
      .oMedia.Play()                    && Начать воспроизведение
   ENDWITH
ENDIF
```

В метод `Click` кнопки **Играть** введите код из листинга 24.16.

```
WITH this.Parent
   .oMedia.Play()
   .Timer1.Enabled = .t.
ENDWITH
```

В метод `Click` кнопки **Пауза** введите код из листинга 24.17.

```
WITH this.Parent
   .oMedia.Pause()
   .Timer1.Enabled = .f.
ENDWITH
```

В метод `Click` кнопки **Стоп** введите код из листинга 24.18.

```
WITH this.Parent
   .oMedia.Stop()
   .Timer1.Enabled = .f.
   .Interval.value = 0
ENDWITH
```

Вот и все, что нужно сделать для того, чтобы управлять воспроизведением звукового файла. Теперь напишем код отображения процесса в шкале управляющего элемента **Slider Control**, которому мы дали имя **Interval**.

Управлять положением ползунка, естественно, будем из таймера. Его событие `Timer` возникает два раза в секунду (потому что мы так решили), что обеспечивает вполне плавное перемещение ползунка по шкале.

Код метода `Timer` таймера приведен в листинге 24.19.

```
WITH this.Parent
   IF .oMedia.GetSize() <= .oMedia.GetPosition() && Конец файла?
```

```
      .cmdStop.Click()                          && Да, останавливаемся
   ELSE                                         && Нет,
      .interval.value = .oMedia.GetPosition()   && показать позицию
   ENDIF
ENDWITH
```

В методе проверяется, не достигнут ли конец файла. Если нет, то запрашивается текущая позиция воспроизведения и ее значение присваивается свойству `Value` слайдера, тем самым выполняется перемещение ползунка. Если же файл закончился, то эмулируется нажатие на кнопку **Стоп**.

Для того чтобы при перемещении ползунка мышью мы могли перейти к нужному фрагменту в воспроизводимом файле, нужно обрабатывать событие `Change` объекта `Interval`, которое возникает в момент "отпускания" мышью ползунка. Код метода `Change` приведен в листинге 24.20.

```
WITH this.Parent
   .oMedia.SetPosition(this.value)  && Перейти на  новую позицию
   = INKEY(0.05)                    && Задержка на 50 миллисекунд
   .cmdPlay.Click()                 && Продолжить воспроизведение
ENDWITH
```

Обратите внимание на то, что в метод вводится искусственная задержка на 50 миллисекунд. Эта задержка необходима при воспроизведении очень больших файлов (десятки и сотни мегабайт) для того, чтобы MCI успела выполнить позиционирование до получения следующей команды. Конечно, можно было бы организовать бесконечный цикл, в котором проверять готовность устройства, но предложенное решение, на наш взгляд, гораздо проще и эффективнее.

В момент "захвата" ползунка мышью таймер необходимо остановить. Для этого в метод `MouseDown` слайдера добавьте следующий код:

```
LPARAMETERS button, shift, x, y
this.Parent.Timer1.Enabled = .f.
```

Последнее, что нам осталось сделать, — это написать код для регулировки уровня звука. Для этого в метод `Scroll` управляющего элемента **Slider Control** с именем **Volume** введите следующий код:

```
this.Parent.oMedia.SetVolume(this.value)
```

И все! Наш проигрыватель готов к работе. Создайте форму, разместите на ней класс `vfpplayer`, запустите форму на выполнение и наслаждайтесь музыкой!

Проигрыватель видеофайлов

Создайте новую форму, разместите на ней объект — экземпляр класса `vfpplayer` (Конструктор присвоит ему имя **vfpplaier1**). В верхней области формы разместите

объект `Shape`. Нам понадобятся значения его свойств `Left`, `Top`, `Width` и `Height` для определения области вывода видео. Залейте этот `Shape` черным цветом, чтобы обозначить область экрана. В Конструкторе форм эта форма должна выглядеть так, как показано на рис. 24.5.

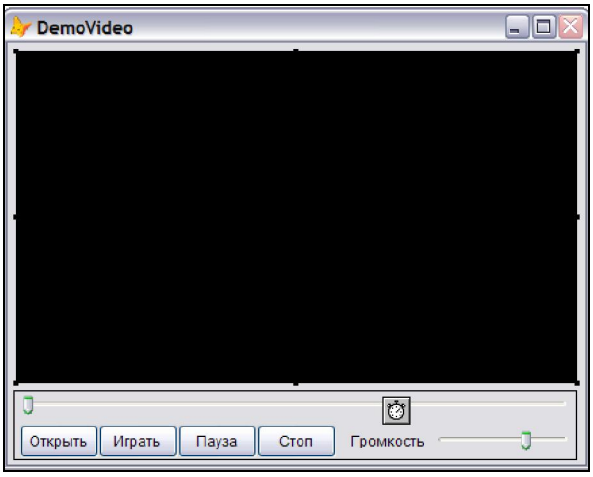

Рис. 24.5. Форма для воспроизведения видео в Конструкторе форм

Откройте метод `Click` кнопки `cmdOpen` контейнера `vfpplayer1` формы на редактирование и введите в него следующий код (листинг 24.21).

```
LOCAL lcFile, lnDestWidth, lnDestHeight, lnKVert, lnKHor, lnKoeff, ;
      lnTop, lnLeft, lnWidth, lnHeight,
lcFile = GETFILE('AVI')
IF !EMPTY(lcFile)
   WITH this.Parent
      .oMedia.Open(lcFile, thisform.HWnd)
      .oMedia.SetVolume(750)
      .Interval.value = 0
      .Interval.max = .oMedia.GetSize()
* Получить размеры в виде кадра
      lnDestWidth = 0
      lnDestHeight = 0
      .oMedia.GetVideoDimension(@lnDestWidth, @lnDestHeight)
* Определение фактических размеров области вывода видео
      lnKVert = lnDestHeight / thisform.Shape1.Height
      lnKHor = lnDestWidth / thisform.Shape1.Width
      lnKoeff = MAX(lnKVert, lnKHor)
      IF lnKoeff < 1
         lnKoeff = 1
      ENDIF
```

```
      lnWidth = lnDestWidth / lnKoeff
      lnHeight = lnDestHeight / lnKoeff
* Вычисляем новые значения координат левой верхней точки
      lnTop = thisform.Shape1.Top+0.5*(thisform.Shape1.Height-lnHeight)
      lnLeft = thisform.Shape1.Left+0.5*(thisform.Shape1.Width-lnWidth)
* Устанавливаем новые размеры окна для видео
      .oMedia.SetVideoRect(lnLeft, lnTop, lnWidth, lnHeight)
      .oMedia.Play()
      .Timer1.Enabled = .t.
   ENDWITH
ENDIF
```

Сохраните форму, например, с именем **DemoVideo**. Запустите ее на выполнение, нажмите на кнопку "Открыть" и выберите в диалоговом окне **Open** видеофайл. Изображение должно вписываться в область, занимаемую элементом `Shape`.

> ***Замечание***
>
> Как уже говорилось, MCI автоматически распознает форматы сжатия *mpeg* и *h261*. Если вы загрузили файл, использующий другой формат, то фильм может воспроизводиться неверно или не воспроизводиться вообще. Узнайте, какой формат сжатия использует этот файл, и передайте его название в команде `SETVIDEO` *алиас* `Algorithm` *значение*.

Microsoft Agent

Microsoft Agent — это интересная технология создания интерактивных анимированных персонажей в приложениях Windows и на страницах Интернета.

В приложениях Microsoft Office уже давно существует персонаж, который постоянно вам что-то подсказывает, двигается и выполняет множество различных действий. Microsoft Agent — это такой же персонаж, и вы можете задействовать его в своих приложениях. Он сможет двигаться, перемещаться по экрану, а если к приложению долго не обращаться, то может даже уснуть. А как вы относитесь к тому, что персонаж Microsoft Agent умеет говорить, да еще и по-русски?

Что же мешает программистам более активно использовать технологию Microsoft Agent в своих разработках? Скорее всего, незнание этого компонента, хотя он входит в состав Windows 2000/ME/XP, а на сайте Microsoft доступен бесплатный редактор для создания новых персонажей; более того, в Интернете появляется все больше сайтов, на которых представлены большие коллекции персонажей.

В папке Ffc, расположенной в корневой папке Visual FoxPro, вы найдете библиотеку классов _agent.vcx, и в ней — класс _Agent, предоставляющий интерфейс для создания и управления персонажами Microsoft Agent. К сожалению (или, может быть, лучше сказать "как обычно"), в справочной документации к Visual FoxPro описанию этого класса посвящено всего два небольших раздела, которые, к сожалению, практически не дают никакой информации о работе с этим компонентом. Поэтому мы начнем его изучение с самого начала.

Объектная модель Microsoft Agent

Microsoft Agent состоит из следующих объектов:

- Request;
- Agent (control);
- Characters (коллекция объектов Character);
- Character;
- Commands (коллекция объектов Command);
- Command;
- Balloon;
- AnimationNames (коллекция объектов);
- SpeechInput;
- AudioOutput;
- CommandsWindow;
- PropertySheet.

На рис. 24.6 показана иерархическая структура Microsoft Agent. Пунктирные линии возле объектов указывают, что может существовать множество таких объектов.

Ядром Microsoft Agent является объект `Agent`, который обеспечивает доступ к методам и свойствам других объектов, представленных в иерархии. Как и любой другой COM-объект, Microsoft Agent создается функциями `NEWOBJECT()` и `CREATEOBJECT()`:

```
oAgent = CREATEOBJECT('Agent.Control.2')
```

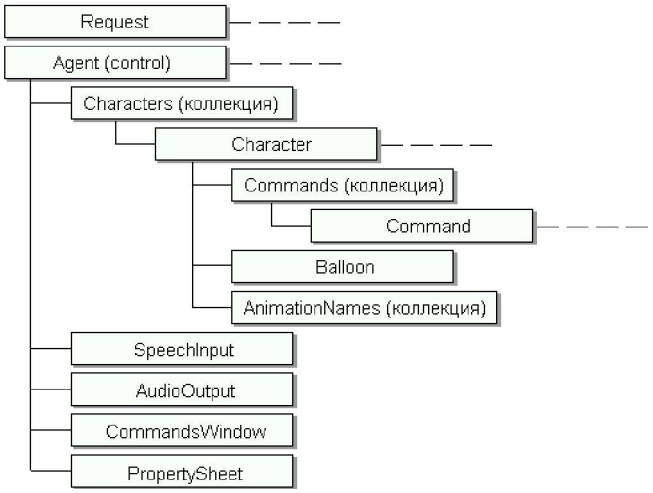

Рис. 24.6. Объектная модель Microsoft Agent

После создания объекта необходимо "сообщить" ему о том, что мы готовы с ним "соединиться":

```
oAgent.connected = .T.
```

После выполнения этой команды объект Microsoft Agent готов к использованию.

Коллекция персонажей *Characters*

Ваше приложение может одновременно использовать одного или нескольких персонажей. Microsoft Agent размещает объекты `Character` (персонаж) в коллекции `Characters` (персонажи).

Метод `Load` объекта `Characters` загружает данные о персонаже в коллекцию персонажей, а также присваивает этому персонажу специальный идентификатор, который используется объектами Microsoft Agent для управления персонажем. Файл персонажа имеет расширение acs. Одного из персонажей (его зовут Мерлин), поставляемого вместе с Windows XP, вы можете загрузить из файла `..\Windows\MSAgent\Chars\Merlin.acs`. Множество различных персонажей вы можете найти и в Интернете, например, на сайте **http://characters.narod.ru/**.

Обращение к методу:

```
oAgent.Characters.Load(CharacterID, Provider)
```

где:

- *`CharacterID`* — символьная строка, определяющая идентификатор персонажа;
- *`Provider`* — символьная строка, определяющая полную спецификацию файла персонажа или адрес персонажа в сети Интернет.

В следующем фрагменте кода показано, как загрузить данные о персонаже Мерлин:

```
oAgent.Characters.Load("MyChar", "C:\Windows\MSAgent\Chars\Merlin.acs")
```

Если загружаемый персонал имеет значок (а правильно сделанный персонал должен всегда иметь значок!), то после загрузки этот значок появится в полосе **TaskBar** (рис. 24.7).

Рис. 24.7. Отображение иконки Microsoft Agent в полосе **TaskBar**

Щелчок правой кнопкой мыши по иконке приводит к появлению контекстного меню, при помощи которого вы можете управлять персонажем, например, скрыть или снова показать.

Метод `Unload` объекта `Characters` удаляет персонажей из коллекции. Методу передается идентификатор персонажа, определенный вами в методе `Load`:

```
oAgent.Characters.UnLoad(CharacterID)
```

Метод `Character` объекта `Characters` возвращает ссылку на объект `Character`:

```
oChar = oAgent.Characters.Character(CharacterID)
```

Это метод используется по умолчанию. Например, две следующих команды дают одинаковый результат:

```
oChar = oAgent.Characters.Character("MyChar")
oChar = oAgent.Characters("MyChar")
```

Использование ссылки на объект внутри коллекции — это очень удобный способ, упрощающий написание кода. Ниже показаны два варианта обращения к методу `Show` объекта `Character`, первый — без использования ссылки, второй — с использованием ссылки:

```
* Без использования ссылки
oAgent.Characters("MyChar").Character.Show()
* С использованием ссылки
oChar = oAgent.Characters("MyChar")
oChar.Show()
```

Естественно, что после завершения работы с Microsoft Agent вы должны уничтожить все созданные объектные ссылки, присвоив ссылочной переменной любое иное значение, например, логическое, либо уничтожив эту переменную командой `RELEASE`.

Объект *Character* (персонаж)

Если выполнить следующие команды:

```
oAgent = CREATEOBJECT('Agent.Control.2')
oAgent.Connected = .t.
oAgent.Characters.Load("MyChar", "C:\Windows\MSAgent\Chars\Merlin.acs")
oChar = oAgent.Characters("MyChar")
oChar.Show()
```

то в верхнем левом углу экрана монитора появится изображение персонажа по имени Мерлин (рис. 24.8).

Рис. 24.8. Microsoft Agent по имени Мерлин

Пока что этот персонаж ничего не умеет делать. Вы можете "зацепить" его мышью и перетащить его в любое место экрана, можете щелкнуть по нему правой кнопкой мыши и в появившемся контекстном меню, состоящем всего из одного пункта

Скрыть, выбрать этот пункт — и персонаж исчезнет с экрана. Научить персонаж двигаться, перемещаться по экрану и даже говорить вы можете при помощи рассматриваемых ниже методов и свойств объекта `Character`.

Метод *GestureAt*

Метод `GestureAt` заставляет персонаж сделать жест рукой в сторону указанной точки:

```
oChar.GestureAt(x, y)
```

Координаты точки задаются в пикселах; используется система координат монитора, причем пиксел с координатами 0, 0 находится в левом верхнем углу экрана. Поэтому если вы хотите, чтобы персонаж показывал на определенный объект, расположенный на вашей форме, то к координатам верхней левой точки этого объекта необходимо добавить значения свойств `Top` и `Left` формы.

Метод *Hide*

Метод `Hide` скрывает персонаж (персонаж исчезает с экрана), причем режим исчезновения передается методу как параметр:

```
oChar.Hide(fact)
```

Если *fact* = `.T.`, то персонаж исчезнет мгновенно. В противном случае вы увидите короткую анимацию, в конце которой персонаж исчезнет.

Для восстановления персонажа щелкните правой кнопкой мыши по его иконке в панели **TaskBar** и в появившемся меню выберите пункт **Показать**.

Метод *MoveTo*

Метод `MoveTo` перемещает персонаж в заданную точку экрана монитора:

```
oRequest = oChar.MoveTo(x, y [, speed])
```

Параметры *x* и *y* — это новые значения координат левого верхнего угла прямоугольной области, в которую вписан персонаж. Необязательный параметр *speed* определяет время перемещения персонажа в миллисекундах. Если параметр не указан, то перемещение персонажа произойдет за одну секунду. Если *speed* = *0*, то персонаж перемещается мгновенно.

> ***ЗАМЕЧАНИЕ***
>
> Напомним, что Microfoft Agent "существует" в мире координат монитора (а не вашей формы). Поэтому если вы хотите позиционировать персонаж внутри формы, то не забывайте прибавлять к координатам персонажа координаты верхнего левого угла формы.

Метод возвращает ссылку на объект `Request`; этот объект позволяет вам отслеживать состояние процесса анимации и управлять им.

Метод *Play*

Метод `Play` заставляет персонаж выполнить одно из предопределенных действий. Перечень действий находится в коллекции `AnimationNames` конкретного персонажа.

Получить список известных персонажу действий можно, выполнив следующий код:

```
nMaxAnimation = 0
DIMENSION aAnimationNames[1]
FOR EACH cAnimName IN oChar.AnimationNames
   nMaxAnimation = nMaxAnimation + 1
   DIMENSION aAnimationNames[nMaxAnimation]
   aAnimationNames[nMaxAnimation] = cAnimName
ENDFOR
```

В результате выполнения кода переменная *nMaxAnimation* будет содержать допустимое количество действий персонажа; наименования действий записаны в элементы массива *aAnimationNames*. Имейте в виду, что каждый персонаж имеет как стандартный набор действий, включающий действия типа `RestPose`, так и действия, специфические для конкретного персонажа.

Список некоторых действий персонажа по имени Мерлин приведен в табл. 24.10.

Таблица 24.10. *Действия Microsoft Agent по кличке Мерлин*

Наименование действия	Описание
Acknowlegle	Кивает головой
Alert	Дергает руками
Announce	Играет на трубе
Blink	Моргает
Confused	Почесывает затылок
Congratulate	Достает кубок
Congratulate_2	Аплодирует
Decline	Разводит руками и отрицательно качает головой

Таблица 24.10 *(окончание)*

Наименование действия	Описание
DoMagic1	Магические движения
DoMagic2	Создает облако дыма
DontRecognize	Прислушивается
Explain	Разводит руками
GestureDown	Показывает вниз
GestureLeft	Показывает влево (от себя)
GestureRight	Показывает вправо (от себя)

GestureUp	Показывает вверх
GetAttention	Стучит в экран один раз
Greet	Кланяется
Pleased	Просит
Process	Варит зелье в котле
Read	Читает книгу
RestPose	Принимает исходное положение (поза при загрузке)
Sad	Вздыхает
Surprised	Радостно вскрикивает "Вау!"
Wave	Прощально машет рукой
Writing	Непрерывно пишет в книгу

Заставить персонаж выполнить действие можно, передав в метод `Play` наименование одного из доступных действий:

```
oRequest = oChar.Play("Greet")
```

Метод так же возвращает ссылку на объект `Request`.

Метод *Show*

Метод `Show` делает персонаж видимым:

```
oChar.Show([flag])
```

Необязательный параметр *flag* определяет, будет ли возникновение персонажа анимированным или персонаж появляется мгновенно. Значение *flag=.T.* означает, что анимация не демонстрируется (персонаж появляется мгновенно).

Метод *ShowPopupMenu*

Метод `ShowPopupMenu` выводит связанное с персонажем контекстное меню:

```
oChar.ShowPopupMenu(x,y)
```

Меню располагается правее и ниже точки, координаты которой передаются в метод как параметры.

Метод *Think*

Метод `Think` выводит над персонажем окно в виде баллона с текстом, как это показано на рис. 24.9.

Рис. 24.9. Текст, отображаемый Персонажем в окне-баллоне

Вот как был вызван метод для получения такого поведения Персонажа:

```
oRequest = oChar.Think("Мне кажется, что я умею думать...")
```

Метод возвращает ссылку на объект `Request`.

Особенностью выполнения метода является то, что текст в окне появляется не мгновенно, а посимвольно.

Метод *Speak*

`Speak` — пожалуй, самый интересный из всех методов объекта `Character`. Как вы уже, вероятно, догадались, это именно тот метод, который позволяет персонажу "произносить речь". Вызов метода:

```
oRequest = oChar.Speak(cText)
```

Передаваемый методу параметр *`cText`* — это строка, содержащая текст, который персонаж будет произносить при помощи синтезатора речи. Одновременно "проговариваемый" текст будет синхронно выводиться в окне-баллоне, расположенном над персонажем.

Для того чтобы персонаж заговорил по-русски, вы должны установить пакет **Text-to-speech engine** для русского языка. Установочные модули **agtx0419.exe** и **lhttsrur.exe** вы можете скачать с сайта, посвященного Microsoft Agent, который расположен по адресу **http://www.microsoft.com/msagent/downloads/user.asp**. После установки этих модулей ваш персонаж заговорит по-русски. Собственно установка **Text-to-speech engine** не вызывает никаких проблем: достаточно запустить указанные файлы на выполнение.

Свойства

Возможно, в некоторых случаях будет необходимо явно указать персонажу, на каком языке он должен говорить. Для этого используется свойство `LanguageID` объекта `Character`.

Идентификаторы некоторых национальных языков приведены в табл. 24.11.

Таблица 24.11. *Коды идентификаторов национальных языков*

LanguageID	Страна	LanguageID	Страна	LanguageID	Страна
0x0405	Чешский	0x0409	Английский (США)	0x0419	Русский
0x0406	Датский	0x0410	Итальянский	0x040B	Финский
0x0407	Немецкий	0x0415	Польский	0x041D	Шведский

Для того чтобы Персонаж заговорил по-русски, необходимо присвоить его свойству `LanguageID` значение 0x0419:

```
oChar.LanguageID = 0x0419
```

В MSDN вы можете найти идентификаторы для тридцати одного языка, на которых могут разговаривать персонажи.

Вот еще некоторые свойства Персонажа, которые вы можете использовать.

Свойства `Width` и `Height` определяют новые значения ширины и высоты Персонажа (в пикселах):

```
oChar.Width = 350
oChar.Height = 200
```

Свойства `OriginalWidth` и `OriginalHeight` возвращают оригинальные значения ширины и высоты персонажа:

```
oChar.Width = oChar.OriginalWidth
oChar.Height = oChar.OriginalHeight
```

Свойство `Visible` позволяет определить, является ли персонал видимым или скрыт. Свойство доступно только для чтения. Если свойство имеет значение `.T.` (истина), то объект отображается на экране.

Управление анимацией. Объект *Request*

Вся анимация выполняется асинхронно. Это означает, что ваша программа не ожидает завершения выполнения действия персонажа. Единственный способ отслеживания процесса анимации — это использование объекта `Request`. Этот объект создается при выполнении методов персонажа, инициирующих процесс анимации, в частности, его создают рассмотренные выше методы `GestureAt`, `Hide`, `MoveTo`, `Play`, `Show` и `Speak`, а также метод `Load` объекта `Characters`.

Объект `Request` имеет свойство `Status`, которое принимает одно из следующих значений (табл. 24.12).

Таблица 24.12. *Значения свойства* `Status` *объекта* `Request`

Значение	Описание
0	Анимация успешно завершена

1	Имела место ошибка при выполнении анимации
2	Выполнение анимации приостановлено
3	Анимация прервана
4	Анимация выполняется (процесс идет)

Вы можете анализировать значение этого свойства и по мере необходимости вызывать описанные ниже методы Персонажа, получающие в качестве параметра ссылку на объект `Request`.

Метод *Stop*

Метод `Stop` прекращает текущую анимацию персонажа и очищает очередь анимаций. Так как Персонаж действует асинхронно, то вы можете направить ему сразу несколько команд, инициирующих различные действия. Это команды образуют очередь; каждая следующая команда выполняется после завершения предыдущей.

В следующем фрагменте кода показан пример использования метода `Stop`:

```
oRequest = oChar.Play("Surprised") && Все действия для персонажа Мерлин
oRequest = oChar.Play("DoMagic1")
oRequest = oChar.Play("RestPose")
* В очередь направлены три анимационных действия
* Выполнение кода вашего приложения
* Следующая команда прерывает процесс анимации и очищает очередь
oChar.Stop(oRequest)
```

Управление воспроизведением речи

К сожалению, речь, синтезированная компьютером, звучит несколько неестественно. Чтобы оживить звучание, заставить персонаж сделать правильную интонацию или громче произнести какое-нибудь слово, нужно использовать специальные управляющие теги, которые вставляются непосредственно в воспроизводимый текст.

Каждый тег ограничивается двумя обратными слэшами; тег может содержать значение, которое отделяется от имени тега символом "равно".

Перечень тегов приведен в табл. 24.13.

Таблица 24.13. *Теги для управления синтезом речи*

Тег	**Вызываемое действие**
`Chr="Normal"`	Обычный голос (используется по умолчанию)
`Chr="Monotone"`	Монотонный голос

Таблица 24.13 *(окончание)*

Тег	Вызываемое действие
`Chr="Whisper"`	Шепот
`Ctx="Address"`	Адрес или телефонный номер
`Ctx="E-mail"`	Адрес электронной почты
`Emp`	Определяет, что на следующем слове делается акцент (ударение)
`Map="spokentext"="balloontext"`	Позволяет отображать один текст, а произносить другой: *Spokentext* — произносимый текст *Balloontext* — текст, отображаемый в окне-баллоне
`Pau=number`	Вставляет в текст паузу. *Number* — целое число, определяющее продолжительность паузы в миллисекундах
`Pit=number`	Устанавливает частоту голоса. *Number* — целое число, определяющее частоту в герцах
`Rst`	Сбрасывает все установки к установкам по умолчанию
`Spd=number`	Устанавливает скорость вывода речи. Нормальная скорость вывода получается при значении *Number* =100
`Vol=number`	Устанавливает громкость речи. *Number* — целое число, которое может принимать значения от 0 до 65 536

Вот как, например, может выглядеть отформатированный текст:

> "\Spd=130\Уважаемый читатель! \pau=500\ Этот \Emp\ пример \pau=200\ написан специально для тебя! \pau=200\ Он показывает \Spd=100\ некоторые мои возможности. \Spd=50\ \Pau=500\ \Chr="Whisper"\ Скажу по секрету: \pau=500\ \Spd=130\ мне очень нравится \pau=200\ говорить по \emp\ русски.\Rst\"

Естественно, форматирующие теги не выводятся в окне-баллоне.

Честно говоря, голосовой синтезатор пока еще несовершенен, и текст приходится форматировать методом экспериментального подбора тегов. Но поверьте, эти мучения того стоят.

> ***Замечание***
> Не все синтезаторы поддерживают полный набор управляющих кодов. Например, в русскоязычном варианте не работает тег "\Pit=*number*\".

Объект *Balloon*

Окно, имеющее вид баллона, располагается над Персонажем Microsoft Agent и предназначено для вывода текста. Процесс вывода текста синхронизирован с произноси-

мыми фразами (по типу караоке). В табл. 24.14 перечислены некоторые свойства этого объекта.

Таблица 24.14. *Свойства объекта* `Balloon`

Свойство	Описание
`BackColor`	Возвращает цвет фона окна в формате RGB (только чтение)
`BorderColor`	Возвращает цвет окантовки окна в формате RGB (только чтение)
`FontCharSet`	Определяет набор алфавитно-цифровых символов
`FontBold`	Используется полужирное начертание шрифта (только чтение)
`FontItalic`	Используется курсив (только чтение)
`FontName`	Наименование шрифта (чтение и запись)
`FontSize`	Высота шрифта (чтение и запись)
`NumberOfLines`	Количество отображаемых в окне строк текста (только чтение)
`Style`	Стиль — целое 32-разрядное число, биты которого определяют стиль окна (чтение и запись)
`Visible`	Логическое значение, управляет видимостью окна (чтение и запись)

В следующем фрагменте кода показано, как установить параметры шрифта для выводимого в окне-баллоне текста:

```
oChar.Balloon.FontName = 'Arial'
oChar.Balloon.FontSize=10
```

В табл. 24.15 перечислены действия, выполняемые при установке битов свойства `Style` окна.

Таблица 24.15. *Битовые маски свойства* `Style` *окна-баллона*

Биты	Описание
0	0 — окно-баллон не появляется; 1 — окно-баллон появляется при вызове методов `Speak` и `Think` объекта `Character`
1	0 — высота окна основана на свойстве `NumberOfLines` объекта; 1 — высота окна подгоняется под размер выводимого блока текста
2	0 — окно остается видимым, пока персонаж не выполнит какое-либо действие или будет перемещен; 1 — окно исчезает после того, как персонаж закончит говорить
3	0 — текст, переданный методу `Speak` объекта `Character`, выводится весь и сразу; 1 — вывод текста синхронизируется с речью; если выводимый текст не помещается в окне, выполняется скроллинг
16—23	Число символов в строке
24—31	Число строк в окне

Объект *AudioOutput*

Этот объект предоставляет следующие доступные только для чтения свойства:

- `Enabled` — если свойство имеет значение "истина", то это означает, что Microsoft Agent поддерживает речевой вывод для персонажей;
- `SoundEffect` — значение "истина" означает, что Microsoft Agent поддерживает звуковые эффекты;
- `Status` — возвращает состояние звукового канала вывода. Его возможные значения приведены в табл. 24.16.

Таблица 24.16. *Значения свойства* `Status` *объекта* `AudioOutput`

Значение	Описание
0	Звуковой канал вывода доступен (не занят)
1	Не имеется никакой поддержки для звукового вывода, например, нет звуковой карты
2	Звуковой канал вывода не может быть открыт (используется другим приложением)
3	Звуковой канал вывода занят, потому что сервер обрабатывает речевой ввод пользователя
4	Звуковой канал вывода занят, потому что Персонаж в настоящее время "говорит"
5	Звуковой канал вывода не занят, но ожидается речевой ввод пользователя
6	Имеется неопознанная проблема при попытке обратиться к звуковому каналу вывода

Объект `AudioOutput` доступен непосредственно из ссылки на Microsoft Agent, созданной функциями `NEWOBJECT` или `CREATEOBJECT`. Ниже показан пример получения значения свойства `Status` объекта `AudioOutput`:

```
nValue = oAgent.AudioOutput.Status
```

ActiveX-компонент Microsoft Agent

Рассмотренный выше способ работы с Microsoft Agent имеет один существенный недостаток, заключающийся в том, что мы не можем перехватывать и обрабатывать события, возникающие при работе с персонажем, например, возникающее при щелчке по персонажу мышью. Решается эта проблема за счет использования ActiveX управляющего элемента Agent. Компонент находится в файле agentctl.dll.

Создайте новую форму и разместите на ней ActiveX управляющий элемент **Microsoft Agent Control 2.0**. Также разместите на форме четыре командные кнопки, **ComboBox** и **EditBox**. Скомпонуйте элементы так, как показано на рис. 24.10.

Добавьте в форму следующие свойства:

- `oChar` — для хранения ссылки на объект `Character` (Персонаж);
- `oRequest` — для хранения ссылки на объект `Request`.

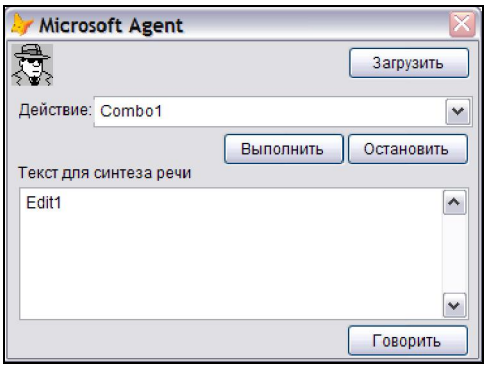

Рис. 24.10. Вид формы Microsoft Agent в Конструкторе форм

Для компонента Microsoft Agent укажите имя **oAgent** (в свойстве `Name`).

В метод `Click` кнопки **Загрузить** введите код из листинга 24.22.

```
LOCAL i, lcFile, lcAnimName
* Выбрать файл Персонажа
lcFile = GETFILE('ACS')
IF EMPTY(lcFile)
   RETURN
ENDIF
* Если существует другой Персонаж, то уничтожаем его
IF VARTYPE(thisform.oChar) = 'O'
   thisform.oAgent.Characters.Unload('Char')
   thisform.oChar = .f.
ENDIF
* Загружаем нового Персонажа в коллекцию Characters
thisform.oAgent.Characters.Load('Char',lcFile)
thisform.oChar = thisform.oAgent.Characters('Char')
* Для объекта Character:
* – формируем список допустимых движений
* – устанавливаем русский язык
* – перемещаем его изображение (пока невидимое) в заданную точку
* – делаем объект видимым
WITH thisform.oChar
   thisform.Combo1.Clear  && Очистить список движений Персонажа
   FOR EACH lcAnimName IN .AnimationNames
      thisform.Combo1.AddItem(lcAnimName)
   ENDFOR
   thisform.Combo1.Refresh
   .LanguageID = 0x0419
   .MoveTo(500,300,0)
   .Show()
ENDWITH
```

При нажатии на кнопку **Загрузить** вы должны в появившемся диалоговом окне **Open** выбрать acs — файл с описанием персонажа, после чего персонаж появится на экране.

В метод `Click` кнопки **Выполнить** введите следующий код (листинг 24.23).

```
IF VARTYPE(thisform.oChar) = "O"
   LOCAL lcOperation
   lcOperation = ALLTRIM(thisform.Combo1.Value)
   IF LEN(lcOperation) > 0
      thisform.oRequest = thisform.oChar.Play(lcOperation)
   ENDIF
ENDIF
```

После загрузки персонажа вы можете посмотреть, как выглядят те или иные его действия. Выберите в выпадающем списке необходимое действие и нажмите на кнопку **Выполнить**.

Некоторые действия персонажа повторяются бесконечно. Поэтому вы должны иметь возможность прервать такое действо. Для этого предусмотрена кнопка **Остановить**. Вот код ее метода `Click` (листинг 24.24).

```
IF VARTYPE(thisform.oChar) = "O"
   thisform.oChar.Stop(thisform.oRequest)
ENDIF
```

Текстовое поле **Edit1** предназначено для ввода текста, который должен произнести персонаж, а кнопка **Говорить** запускает метод `Speak` объекта. Метод `Click` этой кнопки содержит следующий код (листинг 24.25).

```
IF VARTYPE(thisform.oChar) = "O"
   thisform.oChar.Speak(thisform.Edit1.value)
ENDIF
```

Сохраните форму и запустите ее на выполнение. Эффектно, не правда ли?

> ***Замечание***
>
> При вводе текста в поле **Edit1** активно применяйте теги, добиваясь наиболее качественного произношения.

Использование Ассистентов Microsoft Office 2003

В Microsoft Office также используется несколько персонажей. Вы так же можете применять их в своих приложениях, но они не смогут синтезировать речь. То есть все будет работать правильно, но "молча". Связано это с тем, что ассистенты Office и персонажи Microsoft Agent используют разные голосовые движки.

Обработка ошибок Microsoft Agent

Возникающие ошибки, как правило, вызваны недопустимыми командами. Так, если вы не указали имя файла персонажа при его загрузке, то это вызовет исключение и появление сообщения об ошибке. Также ошибка возникает, если вы передаете в метод `Speak` пустую строку (не содержащую никакого текста).

Код ошибки, возникающей во время выполнения анимации, можно узнать, посмотрев значение свойства `Number` объекта `Request`. Описание ошибок Microsoft Agent можно найти в MSDN в разделе *Microsoft Agent Error Codes*.

Заключение

Мир мультимедиа широк и многообразен. Вы познакомились в этой главе лишь с незначительной частью из всего того, что можно применять в ваших приложениях, используя различные функции Windows API. Тем не менее поддержка звука, применение MCI и "разговорчивый" Персонаж Microsoft Agent позволят разнообразить интерфейс вашего приложения, сделав его гораздо более привлекательным.

На прилагаемом к книге компакт-диске вы найдете библиотеку классов для работы с аудио- и видеофайлами, все созданные в этой главе демонстрационные формы, а также небольшую коллекцию Персонажей Microsoft Agent.