ДМИТРИЙ КУЗАН
ВЛАДИМИР ШАПОРОВ

bhv

ПРОГРАММИРОВАНИЕ Win32 API В DELPHI

ВВЕДЕНИЕ В Win32 API

GDI+ – ИНТЕРФЕЙС НОВОГО ПОКОЛЕНИЯ

SIMPLE MAPI – РАБОТА С ЭЛЕКТРОННОЙ ПОЧТОЙ

VIDEO FOR WINDOWS – РАБОТА С ВИДЕО

MEDIA CONTROL INTERFACE – РАБОТА С МУЛЬТИМЕДИА

TAPI – РАБОТА СО СРЕДСТВАМИ КОММУНИКАЦИЙ

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Дмитрий Кузан
Владимир Шапоров

ПРОГРАММИРОВАНИЕ Win32 API в DELPHI

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06
ББК 32.973.26-018.2
К89

Кузан Д. Я., Шапоров В. Н.

К89 Программирование Win32 API в Delphi. — СПб.: БХВ-Петербург, 2005. — 368 с.: ил.

ISBN 5-94157-535-1

Рассмотрено применение различных интерфейсов прикладного программирования Windows (Win32 API) при разработке приложений с использованием Borland Delphi. Описаны основы работы с API. Подробно освещены вопросы практического применения API при создании приложений для работы с электронной почтой (MAPI), со средствами коммуникаций (TAPI), мультимедиа (MMCI), графическим интерфейсом и др. Материал сопровождается наглядными практическими примерами. На компакт-диске расположены исходные тексты примеров, программы и необходимые библиотеки.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Татьяна Лапина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.09.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 29,67.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

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

ISBN 5-94157-535-1

ы

Оглавление

Введение 8

Глава 1. MAPI – интерфейс программирования приложений электронных сообщений 9

Введение 9
Достоинства и недостатки Simple MAPI 11
Подключение Simple MAPI к проекту 12
Отправка сообщения на Simple MAPI 12
Работа с адресной книгой на Simple MAPI 23
Работа с сообщениями на Simple MAPI 28
Коды ошибок Simple MAPI 33

Глава 2. TAPI – интерфейс программирования приложений для работы с телефонией 36

Введение в TAPI 36
Интерфейсы и уровни программирования TAPI 37
Базовый уровень 38
Вспомогательный уровень 39
Расширенный уровень 40
Работа с устройствами линий 40
Основные шаги работы с телефонией 40
Конфигурирование и настройка устройства коммуникации 41
Структура *VarString TAPI* 42
Три механизма уведомлений (сообщений) TAPI 44
Версионность TAPI 46
Определение способностей телефонии 47
Открытие устройства линии 49
Дайте мне ваш ID 50
Базовые функции TAPI 50
Вспомогательные функции TAPI 54

Обработка сообщений линии TAPI....58
LineCallback — функция обработки сообщений линии....59
Сообщения линии TAPI....60
Порядок поступления сообщений для входящих и исходящих вызовов....65
Функции и структуры TAPI, связанные с обработкой сообщений....66
Размещение исходящих вызовов TAPI....71
Форматы номеров телефонов в TAPI....71
Ассистент телефонии....73
Функции ассистента телефонии....74
Установление вызова с помощью низкоуровневых функций линии....75
Принятие входящих вызовов....93
Поиск заинтересованного приложения....93
Неизвестный режим носителей....95
Приоритет режимов носителей....96
Обязанности приложения, принимающего входящие вызовы....97
Регламент работы приложения, определяющего режим носителей....98
Принятие входящего вызова....100
Завершение вызова....102
Функции и структуры TAPI, управляющие приемом вызовов....105
Заключение....117

Глава 3. MCI-интерфейс для работы с мультимедиа....118

Введение....118
Интерфейс командных строк и команд-сообщений MCI....118
Командные строки....119
Команды сообщений....120
Типы и драйверы MCI-устройств....121
Классификация MCI-команд....123
Функции и макросы MCI....128
Сообщения MCI....130
Общие флаги для MCI-команд....133
Структуры данных MCI....133
Практика использования....141
Проигрывание wave-файлов....142
Проигрывание MIDI-файлов....147
Звукозапись....150
Проигрывание Audio-CD....153
Проигрывание видеофайлов AVI....154
Коды ошибок MCI....157
Заключение....162

Глава 4. Video for Windows – интерфейс для работы с видео....163

Краткий экскурс....163
Введение в Video for Windows....164
Установка и требования к работе....164

Практика использования ... 165
Открытие файлов AVI ... 165
Получение информации из заголовка файла AVI ... 167
Доступ к потокам ... 170
Получение информации о потоке ... 171
Работа с кадрами. Сохранение отдельных кадров в формат BMP ... 184
Работа с кадрами. Сохранение BMP-файлов в AVI-формат ... 188
Сохранение потоков в отдельных файлах ... 198
Обработка ошибок VFW ... 203
Заключение ... 205

Глава 5. GDI+ — графический интерфейс нового поколения ... 206

Введение в GDI+ ... 206
Установка и требования к работе ... 207
Объектная модель библиотеки ... 208
Первые шаги ... 209
Классы GDI ... 212
Класс *AdjustableArrowCap* ... 212
Класс *Bitmap* ... 213
Класс *BitmapData* ... 213
Класс *Brush* ... 213
Класс *CachedBitmap* ... 213
Класс *CharasterRange* ... 213
Класс *Color* ... 214
Класс *CustomLineCap* ... 214
Класс *EncoderParametr* ... 214
Класс *EncoderParametrs* ... 215
Класс *Font* ... 215
Класс *FontCollection* ... 215
Класс *FontFamily* ... 215
Класс *GDIPlusBase* ... 215
Класс *Graphics* ... 215
Класс *GraphicsPath* ... 216
Класс *GraphicsPathIterator* ... 216
Класс *HatchBrush* ... 216
Класс *Image* ... 217
Класс *ImageAttributes* ... 217
Класс *ImageCodecInfo* ... 217
Класс *InstalledFontCollection* ... 217
Класс *LinearGradientBrush* ... 218
Класс *Matrix* ... 218
Класс *Metafile* ... 218
Класс *MetafileHeader* ... 218
Класс *PathData* ... 218
Класс *PathGradientBrush* ... 218

Класс *Pen* 219
Класс *Point* 219
Класс *PointF* 219
Класс *PrivateFontCollection* 219
Класс *PropertyItem* 219
Класс *Rect* 220
Класс *RectF* 220
Класс *Region* 220
Класс *Size* 220
Класс *SizeF* 220
Класс *SolidBrush* 220
Класс *StringFormat* 220
Класс *TextureBrush* 221
Перечисления GDI+ 221
Константы и структуры GDI+ 222
Практика использования 222
Рисование графических примитивов 222
Работа с изображениями 226
Использование кэшированных растров для повышения производительности вывода 237
Использование кодеров и декодеров изображений 239
Работа со списком кодеков 239
Получение CLSID кодера изображения 240
Определение параметров кодера 241
Сохранение изображений 243
Работа с метаданными 249
Использование Alpha-канала для создания эффектов прозрачности 251
Работа с текстом 254
Координатная система 267
Преобразования (трансформации) объектов 271
Использование регионов 280
Печать 282
Заключение 285

Глава 6. Windows API 286

Типы данных 286
Константы 290
Строки 290
Дескрипторы 292
Сообщения 293
Синтаксис функций Windows API 296
Параметры функций 297
Импортирование функций Windows API 298
Нестандартно импортируемые функции 298
Функции обратного вызова 299

Использование справочной системы по функциям Windows API.....300
Delphi и функции API.....301
Функции управления окнами.....302
Функции ввода/вывода в файл.....314
Функции ввода.....331
Строковые функции и функции атомов.....337
Функции работы с буфером обмена.....343
Функции системной информации.....347
Функции каретки, курсора и иконок.....355
Заключение.....362

Приложение. Описание компакт-диска..... 363

Предметный указатель..... 364

Введение

Уважаемый читатель, вы держите в руках книгу, рассказывающую о взаимодействии Delphi с различными интерфейсами прикладного программирования (API). В ней мы постарались ознакомить вас с различными API и показать работу с ними в Delphi. Нами не ставилась задача подробно и полностью описать каждый интерфейс прикладного программирования, да и это было бы невозможно в рамках одной книги. Скорее всего, ее можно было бы охарактеризовать как вводящую читателя в мир API. Не секрет, что в настоящее время существует огромное количество книг по Delphi, но, к сожалению, большая часть из них посвящена вопросам визуального программирования. Очень часто в них встречаются фразы типа "возьмите компонент Tlabel из палитры компонентов и положите его на форму". Авторы ни в коем случае не против таких книг, но практика показывает, что многие из них просто дублируют друг друга. Стоит также упомянуть, что большинство из них предназначено для новичков. Мы же в свою очередь постарались отойти немного в сторону от стандартов и выбрали тему взаимодействия Delphi с различными API не случайно. Дело в том, что в настоящее время мало материалов, посвященных Delphi и API, к тому же большая их часть опубликована на английском языке, и этой книгой мы постарались хоть как-то заполнить вакуум, присутствующий в компьютерной литературе. Мы прекрасно понимаем, что она вряд ли будет служить полным руководством для разработчиков, использующих соответствующие API, но она может стать, и надеемся, станет первым шагом, который заставит программистов, использующих Delphi, начать изучать различные API. Ведь на Delphi можно создавать не только различные оконные приложения и приложения для работы с базами данных, но и сложные профессиональные приложения, например, для работы со звуком, видео или электронными коммуникациями. Тем более что такие приложения в большинстве используют сторонние API. И каждый уважающий себя программист должен если не знать, то хотя бы разбираться в многообразии различных API.

Итак, мы надеемся, что данная книга послужит путеводителем по различным API для программистов и даст толчок для усовершенствования знаний в программировании. Удачи вам!

ГЛАВА 1

MAPI – интерфейс программирования приложений электронных сообщений

Введение

С каждым днем электронными коммуникациями пользуются все больше и больше людей. Электронный почтовый ящик уже не вызывает удивления, как некоторое время назад; напротив, в компьютерном сообществе все меньше и меньше остается людей, не имеющих электронного почтового ящика. Эта тенденция продолжает сохраняться, поэтому перед программистами встают все новые и новые задачи. Если раньше заказчику требовался просто отчет из базы данных, то теперь заказчик уже хочет, чтоб этот отчет отправлялся ему на дом автоматически по электронной почте. Время не стоит на месте, и смогут ли разработчики не впасть в уныние от множества запутанных стандартов API, которые определяют для себя системы электронных коммуникаций. Фактически основными протоколами передачи электронных почтовых сообщений стали SMTP и POP3, и современный разработчик должен в них разбираться. Тут знающий читатель может задать вопрос: "А зачем мне разбираться с этими протоколами, если в Delphi есть для этого необходимые компоненты, кинув которые на форму я могу придать своему приложению необходимую функциональность?". Это, безусловно, так. Однако на этом пути есть пара, на наш взгляд, очевидных минусов.

Перечислим некоторые из них.

- *Необходимая минимальная настройка* — попробуем объяснить, что мы под этим подразумеваем. Несмотря на все преимущества этих компонентов, их необходимо настроить, т. е. в вашем приложении предусмотреть возможность хранения таких настроек, как электронный адрес, пароль на ящик и пр. Если вы разрабатываете одновременно несколько приложений, использующих электронную почту, то трудоемкость программирования резко возрастает.

- *Отсутствие встроенных средств протоколирования* — один из самых больших недостатков компонентов. Предположим, вы разрабатываете приложение, позволяющее отправлять индивидуальные письма большому количеству получателей автоматически, и вам необходимо предусмотреть возможность ведения архива отправленных сообщений. При определенном опыте можно написать собственные средства ведения логов, однако согласитесь, если бы они были встроены заранее, жить стало бы намного легче.
- *Встроенные ошибки* — несмотря на тщательное тестирование этих компонентов, в них могут быть заложены ошибки. Так сами авторы столкнулись с тем, что стандартные компоненты NMSMTP и NMPOP3, расположенные на вкладке FastNet среды Delphi, не всегда ведут себя корректно. В частности, NMPOP3 не очень хорошо работает с кодировкой KOI8-R.
- *Изменчивость стандартов* — несмотря на довольно широкое распространение стандартов, нет никакой гарантии, что эти стандарты не изменятся в будущем. Соответственно может настать время, когда ваше приложение просто перестанет работать. Хоть это и маловероятно, но возможность существует.

А теперь представим, что в ваши руки попал инструмент электронных сообщений, практически не ограниченный данными минусами. Используя его, вы навсегда забудете заботы о совместимости протоколов и стандартов, все неудобства, связанные с компонентами. Имя этому средству — MAPI или Messaging API. MAPI — интерфейс программирования приложений электронных сообщений предоставит в ваши руки надежный инструмент работы с почтовыми сообщениями. Вкратце мы попробуем обрисовать, что нам может дать использование MAPI в наших приложениях.

- *Простой способ отправки и приема почты* независимо от установленного почтового клиента. Настройки соединения и подключения берутся из самого почтового клиента. Единственное ограничение: почтовый клиент должен поддерживать стандарт MAPI. Хотя ограничением это назвать можно с натяжкой — популярные Outlook Express, Microsoft Outlook и горячо любимый многими The Bat давно уже поддерживают данный стандарт.
- *Протоколирование* — все отправленные письма будут автоматически ложиться в папку "Отправленные" вашего почтового клиента.
- Отсутствие проблем с национальной кодировкой.
- Простой доступ к общей адресной книге. Теперь не надо вести адресную книгу в самих приложениях.
- Гарантия совместимости в будущем.

И это далеко не полный список тех достоинств, что дает нам MAPI. Однако прежде чем приступить непосредственно к разбору процедур и функций, остановимся на архитектуре MAPI. Прежде всего, запомним, что MAPI не является традиционным API, хотя MAPI и представляет разработчикам необходимые функции. Но это только верхняя часть айсберга! На самом деле *MAPI — это архитектура, предоставляющая спецификацию для всех систем электронных сообщений*.

MAPI — это архитектура, основанная на отраслевом стандарте, она не является чьей-либо собственностью, и разрабатывается по сей день множеством различных производителей программ (ISV — independent software vendors), желающих добавить системе электронных сообщений некоторые функции.

В данной главе мы наиболее подробно остановимся на простом варианте MAPI, выпущенном фирмой Microsoft. Он называется Simple MAPI (простой MAPI) и представляет собой обертку для вызовов простейших функций управления почтовыми сообщениями. Simple MAPI позволяет выполнить самые основные действия по рассылке и приему почтовых сообщений. В Simple MAPI входят 12 основных функций и несколько структур данных.

В данной главе мы не будем останавливаться на более позднем варианте MAPI — MAPI 1.0 или Extended MAPI (расширенном MAPI), кардинально отличающемся от своего предшественника. Подробное рассмотрение MAPI 1.0 вылилось бы в написание отдельной книги, и поэтому мы решили остановиться на Simple MAPI, как на наиболее простом в изучении и понимании. Для тех же, кому нужно получить информацию по использованию MAPI 1.0, а также заголовочные файлы и примеры, мы можем посоветовать обратиться на соответствующие сайты в Интернете, в частности на **www.iCodeTeam.net, www.evocorp.com**.

Однако прежде чем приступить к рассмотрению Simple MAPI, мы немного расскажем о MAPI 1.0. В отличие от Simple MAPI, потомок MAPI 1.0 предоставляет большую функциональность вкупе с широкими возможностями. Основанный на технологии COM данный MAPI 1.0 представляет разработчикам большой спектр функций и COM-интерфейсов для создания приложений электронных коммуникаций. Более сложный в освоении MAPI 1.0 на самом деле является мощным инструментом, позволяющим создавать такие части почтовых систем, как, например, почтовые шлюзы и даже почтовые серверы.

Достоинства и недостатки Simple MAPI

Как вы уже знаете, Simple MAPI предоставляет разработчику простой набор средств для работы с почтой. Данный набор средств идеально подходит для создания небольших прикладных приложений, включающих в себя стандарт-

ные средства работы с почтой. Однако использование Simple MAPI накладывает на разработчика и ряд ограничений, не позволяющих создавать сложные почтовые системы.

- Разработчик не имеет возможности манипулировать сообщениями, находящимися за границами стандартных папок "Входящие", "Исходящие". То есть если вы хотите получать, например, доступ к папке "Письма для Васи", то Simple MAPI такой возможности вам не предоставит.
- Simple MAPI работает только со стандартными полями сообщений, такими как "Тема", "Отправитель", "Получатель" и т. п.
- Simple MAPI представляет собой интерфейс для отправки и приема сообщений и ни на что другое не способен.

Однако Simple MAPI дает простую и четкую возможность с минимальными усилиями встроить поддержку почтовых сообщений в собственное приложение.

Подключение Simple MAPI к проекту

Для того чтобы разработчик Delphi получил в свои руки все возможности Simple MAPI, ему необходимо подключить в секцию `Uses` файл `Mapi.pas`. Данный файл представляет собой заголовочный файл `Mapi.h`, переведенный в синтаксис Object Pascal компанией Borland.

Отправка сообщения на Simple MAPI

Прежде чем приступить к детальному освещению вопросов, касающихся использования Simple MAPI, приведем небольшой пример, позволяющий отправлять почтовые сообщения одному адресату через почтовый клиент пользователя с использованием Simple MAPI. На рис. 1.1 представлен внешний вид демонстрационного приложения, а в листинге 1.1 — его исходный код. Полную версию программы вы сможете найти на прилагаемом компакт-диске в каталоге Source\Ch01\Ex01.

Листинг 1.1. Отправка почтового сообщения с помощью Simple MAPI

```
procedure TFormMain.SendEMail;
var
  MapiMessage: TMAPIMessage;     // Структура содержит информацию
                                 // о посылаемом или
                                 // принимаемом сообщении.
  Receip     : TMAPIRecipDesc;  // Структура получателей.
```

```
  MAPI_Session: Cardinal;       // Сессия
  Flag       : Integer;
  dwRet      : Cardinal;        // Возвращаемое значение
  WndList: Pointer;
begin
  // Открываем сессию Simple MAPI
  dwRet := MapiLogon(Application.Handle,PChar(''),PChar(''),
                   MAPI_LOGON_UI or MAPI_NEW_SESSION,0,@MAPI_Session);
  // Если процесс установки почтовой сессии завершился неудачей
  if (dwRet <> SUCCESS_SUCCESS) then
  begin
      ShowMessage('Не могу установить сессию !');
  end
  else
  begin
     // Обнуляем структуру сообщения и структуру получателей.
     FillChar(MapiMessage, SizeOf(MapiMessage), #0);
     FillChar(Receip, SizeOf(Receip), #0);
     // Заполняем поле "Получатель"
     Receip.ulReserved   := 0;
     Receip.ulRecipClass := MAPI_TO;  // Кому
     Receip.lpszName     := StrNew(PChar(EditTo.Text));
     Receip.lpszAddress  := StrNew(PChar('SMTP:' + EditTo.Text));
     Receip.ulEIDSize := 0;
     MapiMessage.nRecipCount := 1; // Кол-во получателей - один
     // Связываем получателей с сообщением
     MapiMessage.lpRecips := @Receip;
     // Вложенных файлов в сообщении пока нет
     MapiMessage.nFileCount := 0;
     MapiMessage.lpFiles    := nil;
     // Заполняем поле "Тема сообщения"
     if EditSubject.Text <> '' then
        MapiMessage.lpszSubject := StrNew(PChar(EditSubject.Text));
     // Заполняем сам текст сообщения
     if MemoBody.Text <> '' then
        MapiMessage.lpszNoteText := StrNew(PChar(MemoBody.Text));
     WndList := DisableTaskWindows(0);
     try
       IF CheckBoxSend.Checked then
          Flag := 0
       else
          Flag := MAPI_DIALOG; // Показывать стандартное окно почтовика
          MapiSendMail(MAPI_Session, Application.Handle,
                       MapiMessage, Flag, 0);
```

```
    finally
      EnableTaskWindows( WndList );
    end;
    // Очищаем выделенную память
    if Assigned(MapiMessage.lpszSubject) then
       StrDispose(MapiMessage.lpszSubject);
    if Assigned(MapiMessage.lpszNoteText) then
       StrDispose(MapiMessage.lpszNoteText);
    if Assigned(Receip.lpszAddress) then
       StrDispose(Receip.lpszAddress);
     if Assigned(Receip.lpszName) then
       StrDispose(Receip.lpszName);
     // Завершаем сессию
     MapiLogOff(MAPI_Session, Application.Handle, 0, 0);
  end;
end;
```

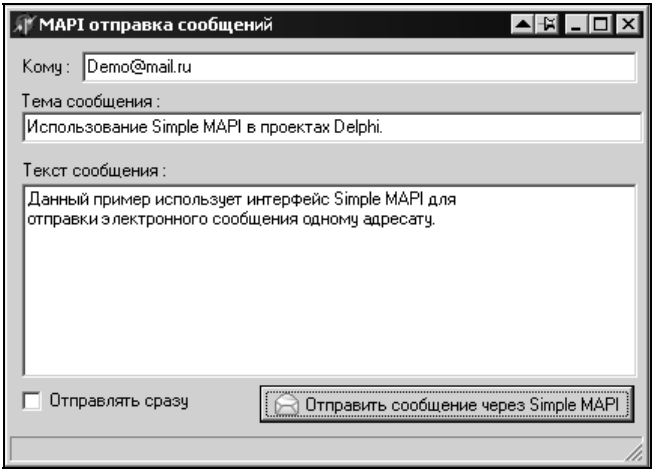


Рис. 1.1. Отправка сообщения через Simple MAPI

Как вы сами видите, для того чтобы отправить сообщение средствами Simple MAPI нужно вызвать всего несколько функций данного интерфейса, а достигнутая функциональность с лихвой окупает затраты на написание исходного кода. После того как мы с вами рассмотрели демонстрационный пример, углубимся в теоретическую часть и для начала определимся с сессиями Simple MAPI.

Концепция работы интерфейса Simple MAPI и прикладной программы основана на использовании сессий или, иначе говоря, соединений между прикладной программой и низкоуровневой системой электронных сообщений.

То есть перед непосредственным использованием функций Simple MAPI по работе с почтовыми сообщениями вы должны установить сессию (сеанс) работы с Simple MAPI с помощью функции `MAPILogon`. Установка сессии происходит в момент регистрации пользователя, когда операционная система убеждается в необходимых правомочиях системы сообщений, а также в отсутствии каких-либо ошибок со стороны профилей службы сообщений.

При установке сессии программист может указать один из двух типов сессии. Сессии могут быть как индивидуальные, так и разделяемые. Индивидуальные сессии являются соединениями "один к одному" между конкретным приложением и Simple MAPI, и не могут быть использованы другими внешними приложениями. Разделяемые сессии, напротив, могут быть использованы сторонними приложениями.

Для создания новой сессии, первичной загрузки адресной книги и списка сообщений нужно использовать функцию `MAPILogOn`. Ее прототип в Object Pascal имеет следующий вид:

```
function MapiLogOn(ulUIParam: Cardinal; lpszProfileName: LPSTR;
          lpszPassword: LPSTR; flFlags: FLAGS;
          ulReserved: Cardinal;
          lplhSession: PLHANDLE):Cardinal;
```

Данная функция имеет следующие параметры, представленные в табл. 1.1.

Таблица 1.1. *Параметры функции* `MAPILogOn`

Параметр	Описание
`ulUIParam`	Дескриптор окна, вызвавшего `MAPILogOn`. Если указан 0, то параметр `ulUIParam` игнорируется
`lpszProfileName`	Указатель на строку, содержащую путь к файлу настроек
`lpszPassword`	Указатель на строку, содержащую пароль для доступа
`flFlags`	Маска флагов дополнительных опций. В данном параметре могут быть установлены следующие флаги: `MAPI_FORCE_DOWNLOAD` — если флаг установлен, то функция пытается загрузить все сообщения на почтовом ящике перед окончанием работы данной функции. Если флаг не установлен, тогда сообщения будут загружены после завершения работы функции; `MAPI_LOGON_UI` — показывает диалог идентификации пользователя, при затребовании системой; `MAPI_NEW_SESSION` — флаг указывает функции сделать попытку создания новой сессии, перед тем как использовать существующую

Таблица 1.1 *(окончание)*

Параметр	Описание
ulReserved	Данный параметр не используется и зарезервирован на будущее
lplhSession	Указатель на сессию Simple MAPI. При создании новой сессии в данный параметр заносится указатель на созданную сессию

После создания сессии все следующие операции с почтовыми сообщениями используют идентификатор сессии, заданный в параметре `lplhSession`. Стоит также отметить, что после окончания работы с Simple MAPI следует закрыть открытую сессию, используя функцию Simple MAPI — `MAPILogOff`:

```
function MapiLogOff(lhSession: LHANDLE; ulUIParam: Cardinal;
          flFlags: FLAGS;
          ulReserved: Cardinal): Cardinal;
```

Данная функция закрывает открытую сессию с почтовой системой и имеет следующие параметры, представленные в табл. 1.2.

Таблица 1.2. *Параметры функции* `MAPILogOff`

Параметр	Описание
lhSession	Указатель на открытую ранее сессию Simple MAPI
ulUIParam	Дескриптор окна, вызвавшего `MAPILogOff`. Если указан 0, то данный параметр игнорируется

Параметры `flFlags` и `ulReserved` не используются и зарезервированы на будущее.

Следующим шагом после открытия сессии стал шаг заполнения полей структуры `TMAPIMessage`, содержащей информацию о посылаемом или принимаемом сообщении. Данная структура в Mapi.pas представлена следующим образом:

```
PMapiMessage = ^TMAPIMessage;
{$EXTERNALSYM MapiMessage}
MapiMessage = packed record
 ulReserved: Cardinal;
  lpszSubject: LPSTR;
  lpszNoteText: LPSTR;
  lpszMessageType: LPSTR;
  lpszDateReceived: LPSTR;
  lpszConversationID: LPSTR;
```

```
  flFlags: FLAGS;
  lpOriginator: PMapiRecipDesc;
  nRecipCount: Cardinal;
  lpRecips: PmapiRecipDesc;
  nFileCount: Cardinal;
  lpFiles: PMapiFileDesc;
 end;
TMAPIMessage = MapiMessage;
```

Структура имеет следующие поля, представленные в табл. 1.3.

Таблица 1.3. *Поля структуры* `TMAPIMessage`

Поле	Описание
`ulReserved`	Поле зарезервировано. Значение поля всегда установлено в ноль
`lpszSubject`	Указатель на строку, содержащую тему сообщения
`lpszNoteText`	Указатель на строку текста сообщения
`lpszMessageType`	Указатель на строку, указывающую тип сообщения
`lpszDateReceived`	Указатель на дату сообщения в формате YYYY/MM/DD HH:MM, где HH (часы) — имеет диапазон от 0 до 24. Данный параметр используется при получении сообщений
`lpszConversationID`	Указатель на идентификатор потока, в который поступило сообщение. Данный параметр используется при получении сообщений
`flFlags`	Маска флагов дополнительных опций сообщения, в данном параметре могут быть установлены следующие флаги: `MAPI_RECEIPT_REQUESTED` — запрашивать подтверждение получения; `MAPI_SENT` — флаг, указывающий на то, что сообщение было послано; `MAPI_UNREAD` — флаг, указывающий на то, что сообщение не было прочитано
`lpOriginator`	Информация об отправителе сообщения
`nRecipCount`	Количество получателей сообщения, записанных в массиве `lpRecips`
`lpRecips`	Указатель на массив структур `TMAPIRecipDesc`, содержащих в свою очередь информацию о получателях сообщения
`nFileCount`	Количество вложенных в сообщение файлов, содержащихся в массиве `lpFiles`

Таблица 1.3 *(окончание)*

Поле	Описание
lpFiles	Указатель на структуру, содержащую информацию обо всех файлах, приложенных к сообщению

Наконец, после того как заполнили структуру TMAPIMessage, мы можем выполнить непосредственно отправку сообщения, используя функцию MAPISendMail:

```
function MapiSendMail(lhSession: LHANDLE; ulUIParam: Cardinal;
          var lpMessage: TMAPIMessage;
          flFlags: FLAGS;
          ulReserved: Cardinal): Cardinal;
```

Данная функция производит отправку сообщения. Достоинством данной функции является то, что она позволяет производить отправку сообщения с вложенными в него файлами или OLE-объектами, что выгодно отличает ее от аналогичной функции Simple MAPI MAPISendDocuments, которая будет рассмотрена далее. Можно сказать, что MAPISendMail является более мощным вариантом функций отправки сообщений Simple MAPI.

Функция MAPISendMail имеет следующие параметры, представленные в табл. 1.4.

Таблица 1.4. *Параметры функции* MAPISendMail

Параметр	Описание
lhSession	Указатель на открытую ранее сессию Simple MAPI
ulUIParam	Дескриптор окна, вызвавшего MAPILogOff. Если указан 0, то данный параметр игнорируется
lpMessage	Указатель на структуру TMAPIMessage, содержащую сообщение для посылки
flFlags	Маска флагов дополнительных опций отправки. В данном параметре могут быть установлены следующие флаги: MAPI_DIALOG — показывает, что будет выведено стандартное окно отправки сообщения почтового клиента; MAPI_LOGON_UI — показывает диалог идентификации пользователя, при затребовании системой; MAPI_NEW_SESSION — флаг указывает функции сделать попытку создания новой сессии, перед тем как использовать существующую
ulReserved	Параметр зарезервирован

Более простой, но в то же время ограниченной альтернативой функции MAPISendMail является функция MAPISendDocuments, которая позволяет подготовить письмо и вложить в него файлы без указания адресов получателей. При использовании данной функции всегда необходимо участие пользователя:

```
function MapiSendDocuments(ulUIParam: Cardinal;
              lpszDelimChar: LPSTR;
              lpszFilePaths: LPSTR;
              lpszFileNames: LPSTR;
              ulReserved: Cardinal): Cardinal;
```

Операционная система использует именно эту функцию для подготовки письма к отправке при вызове контекстного меню "Отправить>Адресат". Функция имеет следующие параметры, представленные в табл. 1.5.

Таблица 1.5. *Параметры функции* `MAPISendDocuments`

Параметр	Описание
ulUIParam	Дескриптор окна, вызвавшего MAPILogOff. Если указан 0, то данный параметр игнорируется
lpszDelimChar	Указатель на разделитель файлов
lpszFullPaths	Указатель на строку, содержащую полные пути до файлов, которые будут вложены в отправляемое сообщение. При наличии нескольких отправляемых файлов имена файлов должны быть разделены знаком, заданным в параметре lpszDelimChar
lpszFileNames	Указатель на строку, содержащую оригинальные названия файлов, под которыми они будут видны в сообщении
ulReserved	Параметр зарезервирован

Пример подготовки двух документов на отправку:

```
MapiSendDocuments(0,';','d:\Письмо.doc;d:\Договор.doc',
          'Письмо.doc;Договор.doc',0);
```

После того как мы разобрали основные вопросы, касающиеся отправки сообщения, переработаем немного наш первоначальный пример, добавив в него возможность выбора нескольких получателей, а также встроив в наше приложение возможность прикрепления файлов к сообщению.

Непосредственно для реализации нам нужно подробно ознакомиться с двумя структурами Simple MAPI — TMAPIRecipDesc (с ней мы сталкивались в первом примере) и TMAPIFileDesc.

Структура TMAPIRecipDesc представлена в Object Pascal следующим образом:

```
PMapiRecipDesc = ^TMAPIRecipDesc;
MapiRecipDesc = packed record
 ulReserved: Cardinal;
  ulRecipClass: Cardinal;
  lpszName: LPSTR;
  lpszAddress: LPSTR;
  ulEIDSize: Cardinal;
  lpEntryID: Pointer;
 end;
TMAPIRecipDesc = MapiRecipDesc;
```

Она содержит информацию о получателе или отправителе сообщения. Структура имеет следующие поля, представленные в табл. 1.6.

Таблица 1.6. *Поля структуры* TMAPIRecipDesc

Поле	Описание
ulReserved	Поле зарезервировано. Значение поля всегда установлено в ноль
ulRecipClass	Идентификатор типа получателя: MAPI_ORIG — основной получатель сообщения; MAPI_TO — первый получатель сообщения; MAPI_CC — получатель копии сообщения; MAPI_BCC — получатель копии blind copy
lpszName	Указатель на строку, содержащую имя получателя (отправителя)
lpszAddress	Указатель на строку, содержащую E-mail-адрес получателя (отправителя). Значение данного поля очень велико, дело в том, что данное поле, в основном, используется и заполняется системой для входящих сообщений. Для исходящих сообщений программист должен принудительно установить адрес в следующем формате: [тип адреса]:[электронный адрес] где в типе адреса указан протокол отправки, в нашем случае SMTP, например: SMTP:demo@mail.ru
ulEIDSize	Размер в байтах указателя ulEIDSize
lpEntryID	Указатель на идентификатор, используемый Simple MAPI, чтобы установить получателя сообщения

Второй структурой, рассматриваемой нами, является структура TMAPIFileDesc, содержащая информацию о прикрепленных вложениях в сообщении:

```
PMapiFileDesc = ^TMAPIFileDesc;
MapiFileDesc = packed record
 ulReserved: Cardinal;
  flFlags: Cardinal;
  nPosition: Cardinal;
  lpszPathName: LPSTR;
  lpszFileName: LPSTR;
  lpFileType: Pointer;
 end;
TMAPIFileDesc = MapiFileDesc;
```

Структура имеет следующие поля, представленные в табл. 1.7.

Таблица 1.7. *Поля структуры* TMAPIFileDesc

Поле	Описание
ulReserved	Поле зарезервировано. Значение поля всегда установлено в ноль
flFlags	Маска флагов, описывающих тип вложения. В данном параметре могут быть установлены следующие флаги: MAPI_OLE — OLE-объект; MAPI_OLE_STATIC — статичный OLE-объект. Если не установлен ни один флаг, то вложение — простой файл данных
nPosition	Положение файлов-вложений в сообщении. Используется в поле MapiMessage.lpszNoteText
lpszPathName	Указатель на строку, содержащую полный путь к файлу
lpszFileName	Указатель на строку, содержащую имя файла. Данное имя файла увидит получатель сообщения. Может отличаться от имени в lpszPathName, если используется временный файл. Если значение не задано или установлено, как Nil, то используется имя из lpszPathName
lpFileType	Тип файла. Если значение установлено Nil, файл будет воспринят, как не распознанный операционной системой

После небольшого ознакомления модернизируем наше первое приложение. Результат модернизации вы можете наблюдать на рис. 1.2. Полный исходный код расположен на прилагаемом компакт-диске в каталоге Source\Ch01\Ex02.

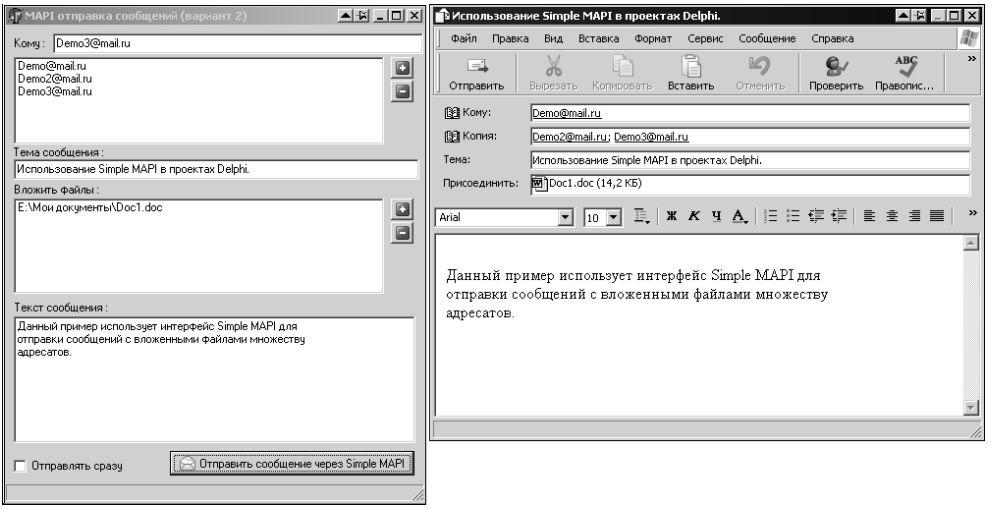


Рис. 1.2. Отправка сообщения

Существенным изменением приложения является то, что в соответствующие поля структуры `TMAPIMessage` передаются не единичные экземпляры соответствующих структур `TMAPIRecipDesc` и `TMAPIFileDesc`, а массивы этих структур, которые и предоставляют возможность множественного добавления получателей и файлов-вложений. Если в первом примере для задания получателя мы передавали просто экземпляр структуры `TMAPIRecipDesc`, то в модернизированном приложении мы передаем массив структур `TMAPIRecipDesc`, выделенных в памяти. Simple MAPI автоматически определяет каждого получателя. Следует также отметить, что в примере при задании множества получателей основным выбирается только один, а все остальные попадают в поле "Копия". Краткий пример создания и работы с массивом получателей представлен в листинге 1.2.

Листинг 1.2. Работа с массивом получателей `TMAPIRecipDesc`

```
type
  // Массив для хранения получателей
  TRecipDescArray = array [0..0] of TMAPIRecipDesc;
  PRecipDescArray = ^TRecipDescArray;
var
  Receip     : PRecipDescArray;    // Указатель на массив получателей.
  ReceipCount: Integer;            // Кол-во получателей
begin
     ReceipCount := ListBoxRecip.Items.Count;
```

```
    if ReceipCount > 0 then
    begin
      // Выделяем область памяти для массива структуры TMAPIRecipDesc
      GetMem(Receip, SizeOf(TMAPIRecipDesc) * ReceipCount);
      For I := 0  to ReceipCount-1 do
      begin
         // Заполняем поле "Получатель"
         RecipName := ListBoxRecip.Items.Strings[I];
         Receip[I].ulReserved   := 0;
         IF I = 0 then
            Receip[I].ulRecipClass := MAPI_TO   // Основной получатель
         else
            Receip[I].ulRecipClass := MAPI_CC;  // Копии

         Receip[I].lpszName     := StrNew(PChar(RecipName));
         Receip[I].lpszAddress  := StrNew(PChar('SMTP:' + RecipName));
         Receip[I].ulEIDSize := 0;
      end;
      MapiMessage.nRecipCount := ReceipCount;
      MapiMessage.lpRecips := @Receip^;
    end

    // Очищаем выделенную память
    // Массивы
    for I := 0 to ReceipCount - 1 do
    begin
        StrDispose(Receip[I].lpszName);
        StrDispose(Receip[I].lpszAddress);
    end;
end;
```

Работа с адресной книгой на Simple MAPI

Для работы с адресной книгой и хранящимися в ней записями в Simple MAPI предусмотрены следующие функции: `MAPIAddress`, `MAPIDetails`, `MAPIResolveName`. Функция `MAPIAddress` создает или модифицирует записи в адресной книге. Данная функция имеет следующий вид:

```
function MapiAddress(lhSession: LHANDLE; ulUIParam: Cardinal;
         lpszCaption: LPSTR; nEditFields: Cardinal;
         lpszLabels: LPSTR;
         nRecips: Cardinal;
         var lpRecips: TMAPIRecipDesc;
```

```
    flFlags: FLAGS;
    ulReserved: Cardinal;
    lpnNewRecips: PULONG;
    var lppNewRecips: PMapiRecipDesc): Cardinal;
```

Параметры функции представлены в табл. 1.8.

Таблица 1.8. *Параметры функции* `MAPIAddress`

Параметр	Описание
`lhSession`	Указатель на открытую ранее сессию Simple MAPI
`ulUIParam`	Дескриптор окна, вызвавшего `MAPIAdress`. Если указан 0, то параметр `ulUIParam` игнорируется
`lpszCaption`	Заголовок окна адресной книги. Если строка пустая, используется название по умолчанию
`nEditFields`	Количество полей редактирования, представленных в адресной книге. Может принимать значения от 0 до 4. При значении поля равном 0 функция предоставляет возможность редактирования записей в адресной книге, остальные значения предоставляют только выбор записей в соответствующие поля "Кому", "Копия" и т. д.
`lpszLabels`	Указатель на строку, используемую в диалоге списка адресов. Если `nEditFields` не равен единице, происходит игнорирование параметра
`nRecips`	Параметр, показывающий, какое число элементов берется из массива `lpRecips`. Если параметр равен нулю, то он игнорируется
`lpRecips`	Указатель на структуру `TMAPIRecipDesc` или на массив структур `TMAPIRecipDesc`. Данный параметр используется для заполнения уже выбранных записей о контактах, которые в свою очередь будут отображаться в соответствующих полях "Кому", "Копия" и т. д., а не в общем списке
`flFlags`	Маска флагов дополнительных опций. В данном параметре могут быть установлены следующие флаги: `MAPI_LOGON_UI` — показывает диалог для идентификации, если это потребуется; `MAPI_NEW_SESSION` — если флаг установлен, функция попытается создать новую сессию, перед тем как использовать существующую
`ulReserved`	Данный параметр зарезервирован
`lpnNewRecips`	Указатель на элемент в массиве `lppNewRecips`. Если параметр равен нулю, то данный параметр игнорируется

Таблица 1.8 *(окончание)*

Параметр	Описание
lppNewRecips	Указатель на структуру TMAPIRecipDesc или на массив структур TMAPIRecipDesc. Данный параметр возвращает записи о контактах, выбранные в адресной книге

Небольшой пример, позволяющий добавлять контакты из адресной книги, представлен в листинге 1.3. Пример максимально упрощен, в частности, мы не стали реализовывать возможность установки уже выбранных контактов в окне адресной книге. Реализацию этого мы оставляем вам для исследования работы функции. Полный исходный текст примера вы сможете найти на прилагаемом компакт-диске в каталоге Source\Ch01\Ex03.

Листинг 1.3. Добавление контактов из адресной книги

```
procedure TFormMain.SpeedButton5Click(Sender: TObject);
var
  dwRet       : Cardinal;
  MAPI_Session: Cardinal;
  I           : Integer;
  lpRecip     : TMAPIRecipDesc;
  intRecips   : ULONG;
  lpRecips    : PMapiRecipDesc;
begin
  // Открываем сессию Simple MAPI
  dwRet := MapiLogon(Application.Handle,PChar(''),PChar(''),
                           MAPI_LOGON_UI or
                           MAPI_NEW_SESSION,0,@MAPI_Session);
  // Если процесс установки почтовой сессии завершился неудачей
  if (dwRet <> SUCCESS_SUCCESS) then
  begin
      ShowMessage('Не могу установить почтовую сессию !');
      SysErrorMessage(dwRet);
  end
  else
  begin
      IF MAPIAddress(MAPI_Session, 0, 'Выбрать пользователей', 1, '', 0,
                     lpRecip,0,0, @intRecips, lpRecips) = SUCCESS_SUCCESS
      Then
      begin
         // Добавляем контакты из адресной книги
         For I := 0 to IntRecips-1 do
```

```
            ListBoxRecip.Items.Add(PMapiRecipDesc(PChar(lpRecips) +
                          I * SizeOf(TMAPIRecipDesc))^.lpszAddress);
        // Очищаем буфер контактов
        MAPIFreeBuffer(lpRecips);
      end;
      // Завершаем сессию
      MapiLogOff(MAPI_Session, Application.Handle, 0, 0);
  end;
end;
```

Функция `MAPIDetails` показывает диалоговое окно, выводящее свойства выбранной записи в адресной книге. Функция имеет вид:

```
function MapiDetails(lhSession: LHANDLE; ulUIParam: Cardinal;
     var lpRecip: TMAPIRecipDesc; flFlags: FLAGS;
     ulReserved: Cardinal): Cardinal;
```

Параметры функции представлены в табл. 1.9.

Таблица 1.9. *Параметры функции* `MAPIDetails`

Параметр	Описание
`lhSession`	Указатель на открытую ранее сессию Simple MAPI
`ulUIParam`	Дескриптор окна, вызвавшего функцию. Если указан 0, то параметр `ulUIParam` игнорируется
`lpRecip`	Указатель на контакт (адресат), для которого надо показать подробности
`flFlags`	Маска флагов дополнительных опций. В данном параметре могут быть установлены следующие флаги: `MAPI_AB_NOMODIFY` — если этот флаг установлен, сведения в диалоговом окне предназначены только для чтения. Пользователь не сможет их изменить; `MAPI_LOGON_UI` — показывает диалог для идентификации, если это потребуется; `MAPI_NEW_SESSION` — если флаг установлен, функция попытается создать новую сессию, перед тем как использовать существующую
`ulReserved`	Данный параметр зарезервирован

Функция `MAPIResolveName` служит для сопоставления имени адресата, внесенного в адресную книгу с электронным адресом, т. е. осуществляет поиск адреса по имени. Функция имеет следующий вид:

```
function MapiResolveName(lhSession: LHANDLE; ulUIParam: Cardinal;
           lpszName: LPSTR;
           flFlags: FLAGS;
           ulReserved: Cardinal;
           var lppRecip: PMapiRecipDesc): Cardinal;
```

Параметры функции представлены в табл. 1.10.

Таблица 1.10. *Параметры функции* `MAPIResolveName`

Параметр	Описание
`lhSession`	Указатель на открытую ранее сессию Simple MAPI
`ulUIParam`	Дескриптор окна, вызвавшего функцию. Если указан 0, то параметр `ulUIParam` игнорируется
`lpszName`	Указатель на строку имени для сопоставления
`flFlags`	Маска флагов дополнительных опций. В данном параметре могут быть установлены следующие флаги: `MAPI_AB_NOMODIFY` — если этот флаг установлен, сведения в диалоговом окне предназначены только для чтения. Пользователь не сможет их изменить; `MAPI_DIALOG` — если указан этот флаг и в адресных книгах найден больше чем один адресат с таким же именем, то будет выведен диалог для выбора имени; `MAPI_LOGON_UI` — показывает диалог для идентификации, если это потребуется; `MAPI_NEW_SESSION` — если флаг установлен, функция попытается создать новую сессию, перед тем как использовать существующую
`ulReserved`	Данный параметр зарезервирован
`lppRecip`	Указатель на структуру `TMAPIRecipDesc`, содержащую сведения об адресате

Помимо основных функций работы с адресной книгой стоит также остановиться на функции `MAPIFreeBuffer`, которую мы использовали в нашем примере.

Данная функция имеет следующий вид:

```
function MapiFreeBuffer(pv: Pointer): Cardinal;
```

Назначение данной функции состоит в том, что она освобождает память, выделенную другими функциями Simple MAPI. В качестве параметра `pv` функ-

ция принимает указатель на буфер памяти. Например, после использования функции MAPIAdress в качестве выбора адресатов система сообщений автоматически выделяет необходимую память для хранения записей в виде массива структур TMAPIRecipDesc. Функция MAPIFreeBuffer призвана корректно освободить занимаемую память во избежание утечки памяти.

Работа с сообщениями на Simple MAPI

Для приема и работы с сообщениями в Simple MAPI реализованы следующие функции:

- MAPIFindNext и MAPIReadMail — для непосредственного приема и чтения сообщений;
- MAPIDeleteMail — для удаления писем;
- MAPISaveMail — для сохранения письма в хранилище сообщений почтового клиента.

Функция MAPIFindNext забирает сообщение, следующее за текущим, и возвращает идентификатор очередного письма из папки "Входящие" (InBox) почтового клиента. Данная функция должна вызываться перед функцией MAPIReadMail, которую мы рассмотрим далее.

Функция MAPIFindNext имеет следующий синтаксис:

```
function MapiFindNext(lhSession: LHANDLE; ulUIParam: Cardinal;
          lpszMessageType: LPSTR;
          lpszSeedMessageID: LPSTR;
          flFlags: FLAGS;
          ulReserved: Cardinal;
          lpszMessageID: LPSTR): Cardinal;
```

Параметры функции MAPIFindNext представлены в табл. 1.11.

Таблица 1.11. *Параметры функции* MAPIFindNext

Параметр	Описание
lhSession	Указатель на открытую ранее сессию Simple MAPI
ulUIParam	Дескриптор окна, вызвавшего функцию. Если указан 0, то параметр ulUIParam игнорируется
lpszMessageType	Указатель на строку-идентификатор класса для поиска
lpszSeedMessageID	Указатель на строку-идентификатор сообщения

Таблица 1.11 *(окончание)*

Параметр	Описание
flFlags	Маска флагов дополнительных опций. В данном параметре могут быть установлены следующие флаги: MAPI_GUARANTEE_FIFO — если флаг установлен, идентификаторы полученных сообщений выводятся в порядке получения (по времени); MAPI_LONG_MSGID — указывает, что идентификатор сообщения превышает 512 символов. При установке данного флага параметр lpszMessageID должен быть больше 512; MAPI_UNREAD_ONLY — указывает на то, что будут выведены только непрочитанные сообщения
ulReserved	Данный параметр зарезервирован
lpszMessageID	Указатель на идентификатор следующего сообщения, т. е. если значение параметра lpszSeedMessageID будет равно 1, то lpszMessageID вернет 2 при условии, что письмо с идентификатором 2 существует

После получения идентификатора сообщения для чтения содержимого нужно вызвать функцию MAPIReadMail, представленную в Simple MAPI в следующем виде:

```
function MapiReadMail(lhSession: LHANDLE; ulUIParam: Cardinal;
         lpszMessageID: LPSTR; flFlags: FLAGS;
         ulReserved: Cardinal;
         var lppMessage: PMapiMessage): Cardinal;
```

Параметры функции представлены в табл. 1.12.

Таблица 1.12. *Параметры функции* `MAPIReadMail`

Параметр	Описание
lhSession	Указатель на открытую ранее сессию Simple MAPI
ulUIParam	Дескриптор окна, вызвавшего функцию. Если указан 0, то параметр ulUIParam игнорируется
lpszSeedMessageID	Указатель на идентификатор сообщения
flFlags	Маска флагов дополнительных опций. В данном параметре могут быть установлены следующие флаги: MAPI_BODY_AS_FILE — показывает, что само сообщение будет сохранено во внешний файл; MAPI_ENVELOPE_ONLY — установка данного флага заставляет Simple MAPI прочитать только заголовок письма;

Таблица 1.12 *(окончание)*

Параметр	Описание
	MAPI_PEEK — при включенном флаге читаемые сообщения не будут помечаться как прочитанные; MAPI_SUPPRESS_ATTACH — установка данного флага не разрешает загружать вложенные файлы, однако само сообщение будет записано в структуру TMAPIMessage
ulReserved	Данный параметр зарезервирован
lppMessage	Указатель на структуру TMAPIMessage

Для демонстрации работы функций MAPIFindNext и MAPIReadMail мы разработали небольшой пример, считающий первое полученное сообщение. Сокращенный исходный код представлен в листинге 1.4. Полный исходный код примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch01\Ex04.

Листинг 1.4. Чтение первого почтового сообщения

```
procedure TFormMain.ReadFirstMail;
const
  RECIP_MAX       = 65000 div SizeOf(TMapiRecipDesc);
type
  // Массив для хранения получателей
  TRecipDescArray = array [0..(RECIP_MAX - 1)] of TMapiRecipDesc;
  PRecipDescArray = ^TRecipDescArray;
var
  PMessage   : PMapiMessage;
  MAPI_Session: Cardinal;         // Сессия
  Str : String;
  Receip     : PRecipDescArray;   // Указатель на массив получателей.
  ReceipCount: Integer;           // Кол-во получателей
  I          : Integer;
  dwRet      : Cardinal;          // Возвращаемое значение
  MessageID: array[0..10000] of Char;  // Номер сообщения
begin
  // Открываем сессию Simple MAPI
  dwRet := MapiLogon(Handle,PChar(''),PChar(''),MAPI_LOGON_UI or
                                    MAPI_NEW_SESSION,0,@MAPI_Session);
  // Если процесс установки почтовой сессии завершился неудачей
  if (dwRet <> SUCCESS_SUCCESS) then
```

```
  begin
      ShowMessage('Не могу установить сессию !');
  end
  else
  begin
     dwRet := MapiFindNext(MAPI_Session, Application.Handle,nil,nil,
                           MAPI_LONG_MSGID OR MAPI_GUARANTEE_FIFO,
                           0,MessageID);
     if (dwRet <> SUCCESS_SUCCESS) then
     begin
          ShowMessage('Ошибка при чтении письма !');
     end
     else
     begin
         IF MapiReadMail(MAPI_Session,Application.Handle,
                         MessageID,0,0,PMessage) = SUCCESS_SUCCESS Then
         begin
            // От
            SetString(Str, PMessage^.lpOriginator^.lpszName,
                      StrLen(PMessage^.lpOriginator^.lpszName));
            EditFrom.Text := Str;

            // Кому
            ListBoxTo.Items.Clear;
            // Определяем кол-во получателей
            ReceipCount:= PMessage^.nRecipCount-1;
            // Передаем указатель
            Receip := @PMessage^.lpRecips^;
            For I := 0 to ReceipCount do
            begin
                SetString(Str, Receip[I].lpszName,
                          StrLen(Receip[I].lpszName));
                ListBoxTo.Items.Add(Str);
            end;
            //* АНАЛОГИЧНО МОЖНО ПОЛУЧИТЬ ДОСТУП И К ВЛОЖЕННЫМ ФАЙЛАМ *

            // Текст сообщения
            SetString(Str, PMessage^.lpszNoteText,
                      StrLen(PMessage^.lpszNoteText));
            MemoBody.Text := Str;
          end;
     end;
     // Завершаем сессию
     MapiLogOff(MAPI_Session, Application.Handle, 0, 0);
  end;
end;
```

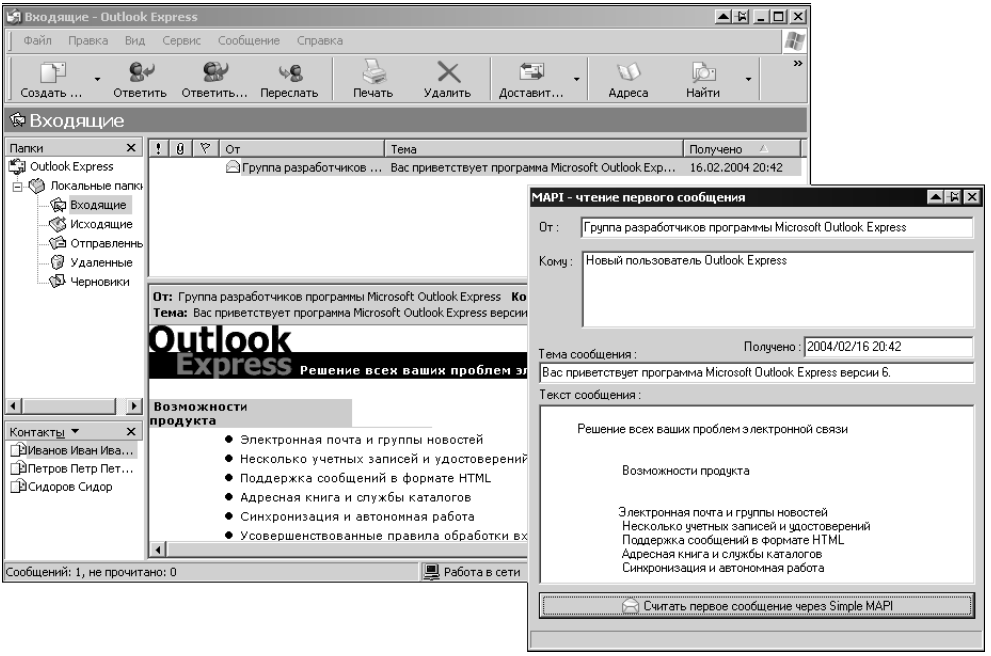


Рис. 1.3. Чтение сообщения через Simple MAPI

Результат работы данного примера представлен на рис. 1.3.

В заключение рассмотрим две оставшиеся функции: функция `MAPISaveMail` сохраняет сообщение в так называемом хранилище сообщений и имеет следующий вид:

```
function MapiSaveMail(lhSession: LHANDLE; ulUIParam: Cardinal;
         var lpMessage: TMAPIMessage;
         flFlags: FLAGS; ulReserved: Cardinal;
         lpszMessageID: LPSTR): Cardinal;
```

Параметры функции представлены в табл. 1.13.

Таблица 1.13. *Параметры функции* `MAPISaveMail`

Параметр	Описание
`lhSession`	Указатель на открытую ранее сессию Simple MAPI
`ulUIParam`	Дескриптор окна, вызвавшего функцию. Если указан 0, то параметр `ulUIParam` игнорируется
`lpMessage`	Экземпляр структуры `TMAPIMessage`. Сообщение, представленное в данной структуре, будет сохранено в хранилище сообщений почтового клиента

Таблица 1.13 *(окончание)*

Параметр	Описание
`flFlags`	Маска флагов дополнительных опций. В данном параметре могут быть установлены следующие флаги: `MAPI_LOGON_UI` — показывает диалог для идентификации, если это потребуется; `MAPI_LONG_MSGID` — этот флаг указывает, что идентификатор сообщения превысит 512 символов; `MAPI_NEW_SESSION` — флаг указывает функции сделать попытку создания новой сессии, перед тем как использовать существующую
`ulReserved`	Данный параметр зарезервирован
`lpszSeedMessageID`	Возвращаемый идентификатор сохраненного сообщения

Функция `MAPIDeleteMail`, как мы говорили ранее, удаляет сообщение. Данная функция представлена в следующем виде:

```
function MapiDeleteMail(lhSession: LHANDLE; ulUIParam: Cardinal;
          lpszMessageID: LPSTR; flFlags: FLAGS;
          ulReserved: Cardinal): Cardinal;
```

Параметры функции представлены в табл. 1.14.

Таблица 1.14. *Параметры функции* `MAPIDeleteMail`

Параметр	Описание
`lhSession`	Указатель на открытую ранее сессию Simple MAPI. Если указан 0, то Simple MAPI создаст временную сессию
`ulUIParam`	Дескриптор окна, вызвавшего функцию. Если указан 0, то параметр `ulUIParam` игнорируется
`lpszMessageID`	Указатель на строку-идентификатор сообщения
`flFlags`	Данный параметр зарезервирован
`ulReserved`	Данный параметр зарезервирован

Коды ошибок Simple MAPI

Для правильной обработки ошибок, возникающих при работе с Simple MAPI, в табл. 1.15 мы приводим список основных ошибок с методами их возможного устранения.

Таблица 1.15. *Ошибки Simple MAPI*

Код ошибки	Описание и способ устранения
SUCCESS_SUCCESS	Функция Simple MAPI была успешно выполнена
MAPI_E_FAILURE	Во время работы функции произошла ошибка, которую Simple MAPI не смог определить
MAPI_E_INSUFFICIENT_MEMORY	Нехватка памяти
MAPI_E_INVALID_EDITFIELDS	Неправильное значение поля nEditFields функции MAPIAddress. Значение поля вышло за отведенный диапазон, заданный от 0 до 4
MAPI_E_INVALID_RECIPS	Ошибка возникает в функциях MAPIAddress, MAPIDetails, если по одному или более контакту (получателю) не найдена информация в адресной книге
MAPI_E_INVALID_SESSION	Неверный хендл сессии, заданный в поле lhSession функции SimpleMAPI
MAPI_E_LOGIN_FAILURE	Неверное имя пользователя. Данная ошибка возникает в функции MAPILogon
MAPI_E_NOT_SUPPORTED	Операция не поддерживается системой сообщений. Очень редкая ошибка, связанная с тем, что установленный по умолчанию почтовый клиент не соответствует требованиям MAPI
MAPI_E_USER_ABORT	Функция прервана пользователем
MAPI_E_INVALID_MESSAGE	Был использован неверный идентификатор сообщения, заданный в поле lpszMessageID. Данная ошибка возникает в функциях работы с сообщениями
MAPI_E_AMBIGUOUS_RECIPIENT	Диалоговое окно не может быть показано. Данная ошибка возникает в функциях MAPIDetails, MAPIResolveName, если значение lpRecip.ulEIDSize равно нулю
MAPI_E_NO_MESSAGES	Данная ошибка возникает в функции MAPIFindNext, если не было найдено ни одного сообщения
MAPI_E_TOO_MANY_SESSIONS	Превышен лимит количества запущенных одновременно сессий. Данная ошибка возникает в функции MAPILogon. Закройте ранее открытие сессии и подключитесь заново
MAPI_E_ATTACHMENT_ WRITE_FAILURE	Вложенный файл не может быть записан. Ошибка возникает в функциях MAPIReadMail, MAPISendMail при неверно заданном пути. Проверьте путь к файлу

Таблица 1.15 *(окончание)*

Код ошибки	Описание и способ устранения
MAPI_E_DISK_FULL	Диск переполнен, функция MAPIReadMail не может сохранить данные
MAPI_E_TOO_MANY_FILES	Превышен лимит на количество вложенных файлов, сообщение не может быть прочитано
MAPI_E_TOO_MANY_RECIPIENTS	Превышен лимит на количество получателей, сообщение не может быть прочитано
MAPI_E_ATTACHMENT_ OPEN_FAILURE	Simple MAPI не удалось найти файл-вложение, заданный в поле lpszFilePaths функции MAPISendMail. Проверьте путь к файлу

ГЛАВА 2

TAPI – интерфейс программирования приложений для работы с телефонией

Введение в TAPI

Бурное развитие телефонных коммуникаций, а также повсеместное использование компьютеров в сферах связи сделало большой прорыв в деле передачи информации по линиям связи. И хотя широко шагающий прогресс преподносит нам все новые и новые виды коммуникаций — от оптоволокна до спутниковой связи — телефон по-прежнему остается самым распространенным средством связи, а наличие модема вкупе с компьютером открывает широкий доступ, начиная от простой передачи факсов и файлов по телефонной линии до выхода в Интернет. В данной главе мы постараемся рассмотреть интерфейс прикладного программирования, специально созданный для управления телефонией для Windows. Имя этому интерфейсу — TAPI.

TAPI — Telephony Application Programming Interface или интерфейс программирования и работы с телефонией в операционной системе Windows. Используя данный интерфейс, вы можете создавать прикладные приложения, поддерживающие широкий диапазон телефонных коммуникаций и разнообразных услуг, использующих телефонную линию. В ваши руки попадет возможность использовать обычную телефонную линию как для передачи данных, так и для передачи речи. Данный интерфейс является неотъемлемой частью операционной системы Microsoft Windows и представляет из себя открытую архитектуру, созданную для упрощения программирования коммуникационных приложений. Высокоуровневые процедуры и функции TAPI скрывают сложности программирования коммуникаций, на низком уровне, позволяя разработчику отбросить в стороны все проблемы, связанные с управлением конкретным коммуникационным устройством, в частности модемом. Однако это не значит, что в TAPI отсутствуют средства низкоуровне-

вого программирования. Также следует отметить, что TAPI позволяет создавать коммуникационные приложения, независимые от установленных устройств коммуникаций, и представляет следующие возможности:

- автоматический набор номера;
- посылка и получение файлов, факсов и электронной почты;
- работа с каналами новостей и информационных услуг;
- возможность управления голосовой почтой;
- автоматизация и обработка поступающих вызовов.

Прежде чем приступить к непосредственному разбору TAPI, мы немного остановимся на взаимодействии Дельфийских приложений с TAPI. Для подключения TAPI в Delphi вы должны поместить в секцию Uses ссылку на файл TAPI.pas, являющийся заголовочным файлом динамической библиотеки TAPI.DLL.

Заголовочный файл TAPI.pas вы можете взять с прилагаемого диска из каталога Install\TAPI Headers\TAPI\.

> ***Примечание***
>
> Каталог Install\TAPI Headers\JEDI Headers Translations\ содержит заголовочные файлы проекта JEDI. В наших примерах данные заголовочные файлы не используются, однако они могут кому-либо понадобится, так что мы их включили в данный диск на всякий случай.

При использовании функций TAPI.pas ваше приложение обращается к TAPI.DLL, которая, в свою очередь, переназначает вызов к TAPI32.DLL. Далее вызов идет к сервису поставщика телефонных услуг (Telephony Service Provider Interface) TAPISRV.EXE, который связывается с конкретными низкоуровневыми драйверами коммуникационных устройств, выполняющими управление, например, HAYES-совместимыми модемами.

Интерфейсы и уровни программирования TAPI

В TAPI реализовано два интерфейса программирования — высокоуровневое и низкоуровневое. Данные интерфейсы предоставляют отдельные функции и структуры для управления коммуникациями. В то время как высокоуровневый интерфейс предлагает горстку функций, упрощающих работу программиста, низкоуровневый интерфейс обеспечивает функциональные дополнительные возможности, а также представляет программисту необходимую гибкость работы с TAPI.

Помимо этого интерфейсы разделены на несколько уровней:

- ❒ базовый (Basic Telephony);
- ❒ вспомогательный (Supplementary Telephony);
- ❒ расширенный (Extended Telephony).

Уровень интерфейсов показывает, насколько тесно приложение, вызывающее функцию, взаимодействует с TAPI.

Стоит заметить, что функции подразделены также на синхронные и асинхронные. Разница между ними состоит в способе обмена данными с коммуникационным устройством. Синхронные функции всегда ожидают ответа на свой запрос, приостанавливая работу приложения; асинхронные ждут ответа, не приостанавливая работу.

После краткого введения мы можем более подробно остановиться на уровнях интерфейсов. Их, как вы знаете, три.

Базовый уровень

На данном уровне находятся функции по спецификации TAPI, которые должны быть совместимы со всеми устройствами коммуникаций, т. е. все устройства коммуникаций, работающие с TAPI, должны полностью поддерживать "базовый" набор функций. Функции этого уровня работают с сервисом POTS.

> ***Примечание***
>
> POTS (Plain Old Telephone Service) — базовый телефонный сервис, основанный на стандартных одноканальных (single-line) телефонах и телефонных линиях.

Стандартную одноканальную схему вы можете увидеть на рис. 2.1.

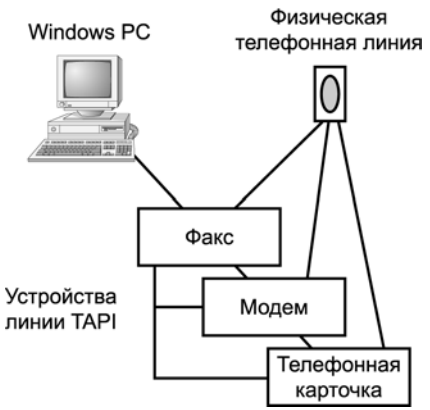


Рис. 2.1. Стандартное одноканальное соединение

Вспомогательный уровень

На вспомогательном уровне находятся "базовые" функции, расширенные дополнительными параметрами или флагами. Расширение функций базового уровня связано с тем, что на данном уровне находятся функции, поддерживающие современные коммуникационные технологии и не вписывающиеся в базовый сервис POTS.

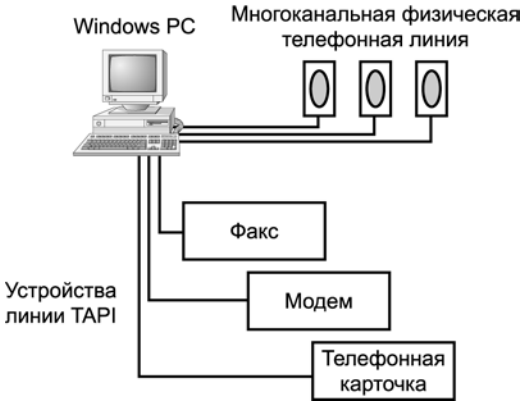


Рис. 2.2. Многоканальные соединения

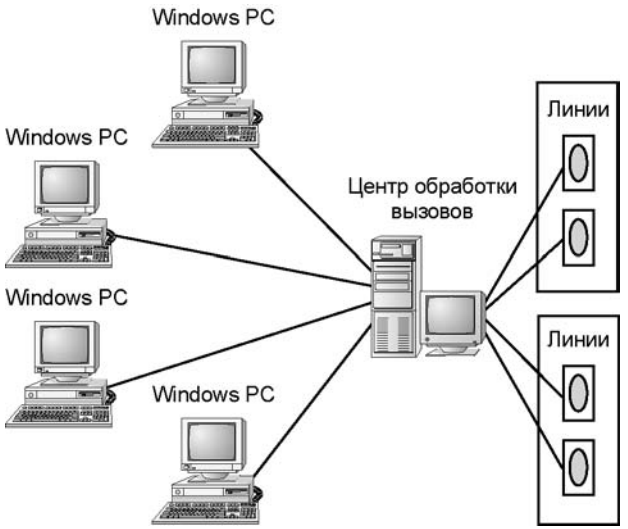


Рис. 2.3. Центр обработки вызовов

Например, на вспомогательном уровне предусмотрена поддержка конференц-связи, многоканальных соединений и офисной телефонной сети (Public

Branch Exchange, PBX). Так, например, структурную схему многоканального соединения вы можете увидеть на рис. 2.2, а на рис. 2.3 — структурную схему центра обработки вызовов. В роли центра обработки может выступать как офисная АТС, так и специализированный компьютер.

Расширенный уровень

В расширенный уровень входят функции, которые являются зависимыми от конкретного устройства коммуникаций. Используя функции из расширенного набора, следует иметь в виду, что ваше приложение становится зависимым от конкретного оборудования. Так как расширенный уровень включает в себя и вспомогательный уровень, это позволяет расширить возможности программы за счет более тесного взаимодействия с конкретным поставщиком услуг или конкретным коммуникационным устройством.

Работа с устройствами линий

Большая часть функций, констант и структур TAPI — это часть API, работающая с так называемыми "устройствами линий". По мере выхода все новых версий TAPI данная часть API увеличивается. В данном разделе мы расскажем об основных операциях с "устройствами линий", такими как инициализация, открытие, закрытие, настройка, а также операциях получения различной статусной информации. Прежде всего, мы рассмотрим понятие "устройства линии". Документация по TAPI трактует устройство линии (или просто "линия") как коммуникационное устройство, связанное с телефоном, т. е. с физической линией. Коммуникационное устройство может быть факсом, модемом или ISDN-картой. Устройства линии обеспечивают широкий диапазон функциональных возможностей телефонии, позволяя вашему приложению посылать или получать разнообразную информацию по телефонной сети.

Следует рассматривать устройства линии не как физическую телефонную линию, а как, скорее всего, логическое представление аппаратных коммуникационных средств вкупе с физической линией связи.

Основные шаги работы с телефонией

Рассматривая работу с телефонией, стоит обратить внимание на то, что последовательность действий очень похожа на работу с другими API, например, с мультимедией. И в том и в другом случае сначала вы должны установить связь с устройством путем его инициализации.

Первичная инициализация TAPI выполняется с помощью функций `lineInitialize` или `lineInitializeEx`. Стоит заметить, что функция

`lineInitialize` является устаревшей и оставлена для совместимости с версией TAPI 1.4. Вместо нее рекомендуется использовать функцию `lineInitializeEx`. Данные функции имеют пять параметров, которые хранят информацию относительно TAPI-сессии, включая адрес функции отзыва (Callback), которая обращается с сообщениями Windows. Новая функция включает два дополнительных параметра, указывающие на структуру `LINEINITIALIZEEXPARAMS`, которая содержит дополнительные параметры, используемые для связи между вашим приложением и TAPI.

После первичной инициализации TAPI и установки связи вы должны предпринять следующие шаги:

1. Проверить, сможет ли TAPI-устройство выполнять заданные вами функции TAPI (если, конечно, данные функции не входят в базовый набор). Данная проверка называется проверкой способностей телефонии (TAPI). Так как не все функции могут обрабатываться конкретным коммуникационным устройством, перед непосредственным выполнением их следует проверять на совместимость. Например, не любой модем может обрабатывать звуковые команды Voice-модема, а только Voice-модем. Для того чтобы проверить способность устройства TAPI, вы должны вызывать следующие функции: `lineGetDevCaps` и `lineGetAddressCaps`.
2. Открыть идентифицированное вами коммуникационное устройство и настроить механизмы сообщений TAPI на ваше приложение. Механизмы сообщений TAPI сродни сообщениям Windows и применяются для информирования приложения обо всех изменениях в линии и в работе коммуникационного устройства. К примеру, определенное сообщение TAPI может информировать вас об обрыве линии.
3. Выполняете определенную работу и закрываете коммуникационное устройство и TAPI.

Далее мы постараемся подробно разобрать каждый шаг.

Конфигурирование и настройка устройства коммуникации

Очень часто пользователям требуется возможность вручную редактировать конфигурацию устройства коммуникации из программы. TAPI представляет программисту для этого две функции — `lineConfigDialog` и `lineConfigDialogEdit`. Обе функции вызывают стандартное диалоговое окно настройки и конфигурирования коммуникационного устройства. Пользователь сам может настроить параметры, связанные с данным устройством. Хоть эти функции и схожи по принципу работы, у них есть существенное разли-

чие: функция lineConfigDialog полностью заменяет конфигурацию оборудования, что может быть чревато последствиями (при неправильном конфигурировании устройства неквалифицированным пользователем можно расстроить работу устройства). Функция lineConfigDialogEdit более демократична, т. к. записывает информацию как бы в буфер (структура VarString) и не вносит изменения сразу. Вы можете позже вызвать функцию lineSetDevConfig и модернизировать конфигурацию на основании записанных в буфер данных.

Также функции позволяют показать любое подокно настройки конфигурации основного диалогового окна. Достигается это использованием параметра lpszDeviceClass. Соответствующие строковые константы окон для lpszDeviceClass вы можете найти в файле помощи TAPI.

Структура *VarString TAPI*

Как упоминалось ранее, функция LineConfigDialogEdit записывает информацию о конфигурации и настройках оборудования в структуру хранения конфигурации. Данная структура VarString определена в TAPI следующем образом (описание структуры взято с перевода TAPI проектом JEDI):

```
PVarString = ^TVarString;
varstring_tag = packed record
dwTotalSize, dwNeededSize,
dwUsedSize, dwStringFormat,
dwStringSize, dwStringOffset: DWORD;
// Modified by Alan C. Moore: new field, next line added
data : array [0..0] of byte;
end;
```

Обратите внимание, что в файле перевода заголовка TAPI от проекта JEDI присутствует дополнительное поле data, где хранятся дополнительные данные об коммуникационном устройстве.

Первые три поля этой структуры — dwTotalSize, dwNeededSize и dwUsedSize — являются обычными полями информации о структуре и часто встречаются в других API Microsoft. Поле dwTotalSize указывает полный размер (в байтах) структуры данных. Вся ответственность по выделению памяти для структур ложится, как мы говорили, на программиста, т. к. в разных версиях TAPI размер структуры может меняться.

Как работать с переменными данными структуры при выделении памяти? Подход заключается в том, чтобы выделить область памяти как бы с запасом под блок переменных данных. Пример в листинге 2.1 покажет вам, как этого можно добиться.

Листинг 2.1. Работа с переменными данными на примере `VarString`

```
if FDeviceConfig=Nil then
begin
   // Выделяем память с запасом в 1000 байт
   FDeviceConfig := AllocMem(SizeOf(VarString)+1000);
   // Обнуляем область памяти
   FillChar(FDeviceConfig^, SizeOf(VarString)+1000, 0);
   // Устанавливаем размер структуры
   FDeviceConfig.dwTotalSize := SizeOf(VarString)+1000;
   // Устанавливаем тип структуры
   FDeviceConfig.dwStringFormat := STRINGFORMAT_BINARY;
end;
```

Поле `dwNeededSize` содержит информацию о размере (в байтах) возвращенной структурой информации. Поле `dwUsedSize` содержит информацию о размере (в байтах) полезной информации, содержащейся в структуре. Данные поля устанавливаются функциями `lineConfigDialog` и `lineConfigDialogEdit`, которые и заполняют данную структуру информацией о конфигурации.

Перед вызовом функции `lineConfigDialogEdit` вы должны вызвать функцию `lineGetDevConfig` и считать начальные данные конфигурации. Эти данные всегда связаны с конкретным коммуникационным устройством (устройством линии). Для обычного модема пользователь, например, может увидеть такие параметры, как `rate`, `character format`, `modulation schemes` и `error control protocol settings`.

Всякий раз, когда вы открываете линию с константой `LINEMAPPER`, вы должны получить идентификатор линии ID с помощью функции `lineGetID`. Данный ID будет затребован во многих функциях TAPI. В нашем случае он нужен в функции `lineGetDevConfig`.

> ***Замечание***
>
> Поскольку структура `VarString` заполняется определенным устройством, может иметь разный размер для каждого устройства, и данные могут быть разными, вы не можете напрямую манипулировать данными, хранящимися в структуре `VarString`. Запомните, данные структуры `VarString` являются данными для внутреннего использования TAPI. Также вы не можете передавать данные о конфигурации одного устройства другому, даже если эти устройства одного класса.

Поговорим подробно об инициализации TAPI.

Как было сказано, установка первичной инициализации TAPI осуществляется функциями `lineInitializeEx` или `phoneInitializeEx`.

Когда ваше приложение вызывает любую из этих функций, TAPI начинает настраивать среду телефонии. Стоит заметить, что на данном уровне не происходит прямого взаимодействия с коммуникационным устройством TAPI. Настройка среды телефонии включает в себя загрузку TAPI.DLL, TAPISRV.EXE и загрузку соответствующих драйверов коммуникационных устройств, указанных в реестре Windows.

Если TAPI решит, что инициализация невозможна, функции могут вернуть следующие коды ошибок: `INIFILECORRUPT`, `LINEERR_NODRIVER`. Первая ошибка заключается в том, что TAPI не может правильно определить настройки телефонии из реестра, вторая заключается в том, что TAPI не смог загрузить соответствующие драйверы коммуникационных устройств или их дополнительных модулей, зарегистрированных в службе телефонии.

Использование функций `lineInitializeEx` или `phoneInitializeEx` всегда должно идти в комплекте с функцией `lineShutdown`, т. е. всегда должен быть реализован механизм открытия-закрытия. Нельзя второй раз открыть TAPI, пока не будет осуществлено закрытие `lineShutdown`. Если по каким-либо причинам вы забудете закрыть TAPI, система телефонии не сможет корректно освободить занимаемые ресурсы.

После успешного завершения функции `lineInitializeEx` возвратятся два параметра, очень важных для последующей работы с телефонией. Это дескриптор TAPI и идентификатор устройства линии (коммуникационного устройства). Как мы кратко упоминали ранее, функция `lineInitializeEx` возвращает в параметре `dwNumDevs` количество коммуникационных устройств, доступных через TAPI. Идентификационные номера устройств лежат в диапазоне от 0 до `dwNumDevs-1`.

Другой критической особенностью TAPI является контроль версий TAPI и способность телефонии. Как мы упоминали в *разд. "Основные шаги работы с телефонией"*, ваше приложение всегда должно проверять способность TAPI корректно выполнить заданные команды. Имейте в виду, что умение TAPI выполнять определенные команды (функции) зависит от многих факторов, таких как конфигурация сети, наличие определенных аппаратных и программных средств (в особенности драйверов), установленных на компьютере, версии TAPI.

Разберем более подробно данные особенности инициализации.

Три механизма уведомлений (сообщений) TAPI

При использовании TAPI-функции `lineInitializeEx` вы должны выбрать один из трех механизмов получения уведомлений (сообщений). Любой из них позволит вашему приложению получать информацию о событиях телефонии.

TAPI представляет следующие:

- Hidden Window — получение уведомлений скрытым окном;
- Event Handle — получение уведомлений внутренним объектом приложения;
- Completion Port — порт завершения или его еще называют комплексный порт.

Далее мы рассмотрим все три механизма, начиная со средств вызова, основных качеств и заканчивая ограничениями, накладываемыми TAPI при использовании данных механизмов. Для того чтобы задать конкретный механизм уведомления в TAPI, вы должны правильно заполнить поле `dwOption` структуры `LINEINITIALIZEEXPARAMS` следующими константами:

- Hidden Window — для включения данного механизма получения сообщений и для совместимости с TAPI 1.X-приложениями используйте константу `LINEINITIALIZEEXOPTION_USEHIDDENWINDOW`;
- Event Handle — `LINEINITIALIZEEXOPTION_USEEVENT`;
- Completion Port — `LINEINITIALIZEEXOPTION_USECOMPLETIONPORT`.

Каждый из трех механизмов получения уведомлений (сообщений) работает по-разному. Так, механизм получения уведомления скрытым окном (Hidden Window) создает скрытое окно, которому будут посылаться сообщения телефонии. Механизм Event Handle создает объект получения сообщений внутри приложения и возвращает дескриптор данного объекта в поле `hEvent` структуры `LINEINITIALIZEEXPARAMS`. Наконец, механизм порта завершения инструктирует TAPI посылать сообщение в ваше приложение, используя функцию `PostQueuedCompletionStatus`. Сразу же хочется обратить внимание на то, что на разработчика ложится вся ответственность за настройку порта завершения. TAPI будет посылать сообщение порту, заданному в поле `hCompletionPort` структуры `LINEINITIALIZEEXPARAMS`. Сообщения будут помечены с ключом завершения, указанным в поле `dwCompletionKey` структуры `LINEINITIALIZEEXLine`.

При выборе механизмов уведомлений стоит учитывать и их возможные ограничения, накладываемые TAPI операционной системой. Мы упомянем только некоторые из ограничений, указанных в файле помощи TAPI. Так, например, если используете механизм получения уведомления скрытым окном, вы должны обеспечить средство считывания сообщений с очереди, причем считывание должно происходить регулярно, дабы избежать отсрочки обработки событий телефонии. Стоит заметить, что хотя основную обработку сообщений для окна Delphi делает автоматически, однако вам придется программировать отзыв на каждое рутинное TAPI-сообщение. Также при вызове `lineShutdown` TAPI автоматически уничтожит созданное окно, поэтому перед

вызовом данной функции вам придется отключать (уничтожать) средство считывания очереди.

Используя механизм Event Handle для получения сообщений, вы попадаете под ограничение, заключающееся в том, что вы не можете напрямую управлять сообщениями TAPI с помощью функций работы с процессами, нитями и волокнами Windows. Например, нельзя использовать функции `SetEvent`, `ResetEvent`, `CloseHandle` и им подобные. Данное ограничение состоит в том, что при использовании данных функций ваше приложение может повести себя самым непредсказуемым образом. Однако допускается использование функций, ожидающих поступления сообщения, вроде `WaitForSingleObject` или `MsgWaitForMultipleObjects`.

Используя механизм Completion Port для получения сообщений, вы должны вручную создавать порт завершения с помощью функции API `CreateIoCompletionPort`. После того как вы создали и настроили порт завершения на получение сообщений, ваше приложение должно считывать события телефонии, поступившие на порт завершения, используя функцию `GetQueuedCompletionStatus`. Данная функция возвратит ключ завершения, заданный в поле `dwCompletionKey` и посланный вашему приложению, а также указатель на структуру сообщения `LINEMESSAGE`, переданную в поле `lpCompletionKey`. После того как ваше приложение обработает данное сообщение, вы должны освободить занятую структурой `LINEMESSAGE` оперативную память. Поскольку ваше приложение создает порт завершения вручную (в отличие от механизма Event Handle), вы должны и закрывать данный порт самостоятельно. Порт следует закрывать после вызова функции `lineShutdown`.

Версионность TAPI

Учитывая то, что TAPI постоянно развивается, остро встает вопрос о совместимости приложений и версий TAPI. К облегчению данный вопрос решается в TAPI просто, на основе так называемого "механизма договоров". Ваше приложение и служба телефонии как бы договариваются друг с другом, какую версию TAPI использовать.

Для чего нужно договариваться о версии TAPI? Дело в том, что существуют несколько различных версий TAPI. Например, TAPI 1.3 создавалась для 16-битных систем. Версия 1.4 изначально была 32-битной и создавалась для Windows 95. Версии 2.x были уже более продвинутыми, но все еще базировались на старом TAPI, ну а в версиях 3.x и более произошла кардинальная переработка TAPI и вывод ее в технологию COM. Помимо этих версий существуют еще версии для мобильных компьютеров, например, для Windows CE (TAPI 1.5). Многообразие TAPI диктует условие, что приложение должно "договориться" со службой телефонии.

Последовательность шагов следующая: при инициализации ваше приложение договаривается о версии с TAPI в два шага.

- В первом шаге устанавливается базовая или TAPI-версия.
- Во втором шаге устанавливается расширенная версия. Базовая версия устанавливается в параметре `dwAPIVersion` функции `lineInitializeEx`.

Во многих простых приложениях TAPI вы можете не договариваться о расширенной версии, однако знайте, что расширенные функции TAPI не будут активированы службой телефонии, и вы не сможете их использовать.

Схему договора можно представить в виде следующего алгоритма. Мы предлагаем системе использовать функции своей версии TAPI, система проверяет, сможет ли она поддержать данные функции и если нет, возвращает версию TAPI, функции которой она сможет поддержать.

Мы поговорили о механизме, но не поговорили о функциях, которые и реализуют данный "механизм договора".

Итак, следующие функции представляют информацию и договариваются о версионности TAPI:

- `lineNegotiateAPIVersion` — для реализации "механизма договора" мы передаем минимальную и максимальную версии TAPI, которые приложение может поддержать в параметрах `dwAPILowVersion` и `dwAPIHighVersion`. После этого TAPI возвратит рекомендуемую версию в поле `dwAPIVersion`, причем возможны ситуации, когда эта версия будет не самой высокой, т. к. TAPI (в частности драйвер устройства) сам определяет номер TAPI-версии, функции которой он может поддержать. Однако номер версии всегда будет возвращен в диапазоне от `dwAPILowVersion` до `dwAPIHighVersion`. Если по каким-либо причинам служба телефонии не может предоставить версию, в данное поле будет возвращен 0;
- для получения информации о расширенной версии следует воспользоваться функцией `lineNegotiateExtVersion`. Данный процесс договора между приложением и TAPI подобен процессу договора функции `lineNegotiateAPIVersion`.

Определение способностей телефонии

При успешном выполнении функции `lineGetDevCaps` и `lineGetAddressCaps` возвратят вам информацию о способностях телефонии на указанном устройстве линии (коммуникационном устройстве). Данная информация будет возвращена в соответствующих структурах данных `LINEDEVCAPS` или `LINEADDRESSCAPS`. Используя информацию, содержащуюся в данных структурах, вы предпринимаете те или иные действия по отношению к заданным

устройствам линии. Стоит также отметить, что по мере выхода новых версий TAPI структуры будут пополняться все новыми и новыми данными. Так как размер структур может быть изменен в разных версиях TAPI, а информация в структуре зависит еще и от устройства, в данных структурах предусмотрено поле `dwNeededSize`, хранящее возвращаемый размер структур. Реальный размер структуры, как и в случае со структурой `VarString`, хранится в поле `dwTotalSize`. Неправильно заданный размер области памяти переменного размера может привести к ошибке `LINEERR_STRUCTURETOOSMALL` при выполнении данных функций. Порядок выделения памяти точно такой же, как и в случае со структурой `VarString`.

Следует заметить, что, используя `lineGetDevCaps` для получения информации о линии, вы получите информацию об адресах линии. Устройство линии может включать множество адресов. Количество адресов будет возвращено в одном из полей структуры `LINEDEVCAPS`. Способности адреса могут измениться так же, как способности линии. Для считывания способностей адреса устройства линии (коммуникационного устройства) вы должны использовать функцию `lineGetAddressCaps` для каждой доступной комбинации `dwDeviceID/dwAddressID`.

Стоит отметить, что дополнительные данные, возвращаемые функциями, записываются в область памяти после области, занимаемой самой структурой `LINEDEVCAPS` или `LINEADDRESSCAPS`. То есть в памяти сначала идет блок, равный `SizeOf(LINEDEVCAPS` или `LINEADDRESSCAPS)` плюс дополнительные данные. В дополнительные данные входит различная информация, не представленная в самой структуре. Например, в данном блоке передается наименование устройства линии. Предвидя резонный вопрос: как считать данную информацию, ответим, что для этого в структурах предусмотрены специальные поля, заканчивающиеся на `Size` и `Offset`. Поясним их назначение, в полях, названия которых заканчивается на Size, хранится информация о размере специфичного блока данных; а в полях, заканчивающихся на Offset, — смещение в байтах от начала структуры. Например, чтобы получить название устройства линии, нужно считать массив данных размером `LineDevCaps.dwLineNameSize`, начиная с блока памяти, равного `PChar(LineDevCaps)` плюс `LineDevCaps.dwLineNameOffset`.

Обратите также внимание на то, что старые приложения (использующие старые версии TAPI, особенно 1.x) не включают подобные поля в состав структур `LINEDEVCAPS` и `LINEADDRESSCAPS`.

Примечание

Так как структуры данных различаются от версии к версии, на программиста ложится задача правильного заполнения полей данных структур.

Для демонстрации определения способностей устройства линии мы разработали небольшую программу, исходный код которой вы сможете найти на прилагаемом компакт-диске в каталоге Source\Ch02\Ex01. Мы не стали подробно останавливаться на раскрытии всех параметров структуры `LINEDEVCAPS` в программе, а сделали как бы небольшой шаблон, дописав который вы сможете получать информацию об устройствах линии.

Открытие устройства линии

Теперь, когда мы разобрались с инициализацией, приступим к шагу, который открывает устройство линии. Как вы помните, устройство линии — это логическое понятие. Поэтому термины "открывая линию", "открывая устройство линии", "работая с линией" или "работая с устройством линии" можно считать взаимозаменяемыми.

Когда ваше приложение откроет линию, вы получите дескриптор данной линии. После открытия линии вам станут доступны все возможности данного устройства линии. В частности, принятие входящих звонков, исполнение типичных задач, посылка и отправка данных и т. д.

Для открытия линии вы должны вызвать функцию `lineOpen`. После окончания работы с линией вы должны ее закрыть функцией `lineClose`.

Открыть линию можно двумя способами: с идентификатором устройства или без него.

Используя первый способ, вызовите функцию `lineOpen` с определенным идентификатором коммуникационного устройства, заданным в параметре `dwDeviceID`. Первый способ откроет именно определенное устройство линии. Если ваше приложение должно обрабатывать входящие вызовы именно с данного коммуникационного устройства, вы должны всегда использовать первый способ. Второй способ заключается в том, что вы еще сами не знаете, с каким коммуникационным устройством будете работать. Вы задаете свойства устройства линии, с которым хотели бы работать, и используете константу `LINEMAPPER` вместо идентификатора устройства в параметре `dwDeviceID`.

Функция откроет любое доступное устройство линии, которое поддерживает определенные вами свойства. Однако открытие линии вторым способом часто может завершаться ошибкой. Но если вы все-таки успешно открыли линию, вы можете определить идентификатор открытого таким способом устройства, вызвав функцию `lineGetID`. Также вам будет известен дескриптор — `lphLine`.

К сожалению, очень редко, но бывают случаи, когда TAPI не может открыть линию. И если такой случай произошел, вы должны определить причину ошибки и постараться ее исправить, если это возможно.

Например, ошибка LINEERR_ALLOCATED указывает на то, что данная линия занята другим приложением.

Дайте мне ваш ID

Функция lineGetID тесно связана с функцией lineOpen. Как было сказано, данная функция возвращает идентификационный номер соответствующего устройства. Принимая дескриптор линии, lineGetID возвратит идентификатор ID открытого устройства линии. Используя данную функцию, вы также можете получить идентификатор различных мультимедийных устройств, таких как Wave, MIDI или Phone. Для чего мы упомянули мультимедийные устройства? Дело в том, что, как вы знаете, TAPI может работать и со звуковыми данными, например, передавать записанный звуковой файл в линию, а такие операции как раз и связаны с мультимедиа.

Базовые функции TAPI

В этом разделе мы кратко остановимся на основных (базовых) функциях TAPI. Как вы знаете, базовые функции TAPI должны поддерживаться всеми совместимыми с TAPI коммуникационными устройствами. В табл. 2.1 кратко представлен список базовых функций, разбитых по категориям. Подробнее о некоторых из них мы расскажем.

Таблица 2.1. *Базовые функции TAPI*

Функция	Описание
Инициализация и закрытие TAPI	
lineInitializeEx	Функция, которая производит первоначальную инициализацию TAPI и подготавливает телефонную линию для использования
lineShutdown	Функция закрывает сессию с TAPI
Статус и свойства устройства линии	
lineNegotiateAPIVersion	Функция позволяет получить сведения о текущей версии TAPI
lineGetDevCaps	Функция возвращает информацию о способностях телефонии на коммуникационном устройстве. Способность телефонии — это возможности, которые вы можете использовать на этом устройстве. Например, может ли это устройство поддерживать голосовые функции

Таблица 2.1 *(продолжение)*

Функция	Описание
`lineGetDevConfig`	Функция возвращает информацию о конфигурации потока коммуникационного устройства
`lineGetLineDevStatus`	Функция, которая возвращает текущий статус телефонной линии, открытой коммуникационным устройством
`lineSetDevConfig`	Функция, которая устанавливает конфигурацию потока на коммуникационном устройстве
`lineSetStatusMessages`	Данная функция позволяет задать, какие сообщения TAPI, связанные с изменением статуса линии и коммуникационного устройства, должны быть получены прикладным приложением
`lineGetStatusMessages`	Данная функция позволяет узнать, какие сообщения TAPI, связанные с изменением статуса линии и коммуникационного устройства, могут получать приложение
`lineGetID`	Функция получает идентификационный номер ID коммуникационного устройства, связанного с открытой телефонной линией, адресом или запросом
`lineGetIcon`	Функция, позволяющая приложению получить иконку соединения для показа ее пользователю
`lineConfigDialog`	Функция, которая вызывает стандартный диалог поставщика телефонных услуг для настройки пользователем параметров коммуникационного устройства, связанного с телефонной линией
`lineConfigDialogEdit`	Аналогична функции `lineConfigDialog`, но более демократична, т. к. записывает информацию о настройках в буфер и не вносит изменения сразу
Работа с адресами	
`lineGetAddressCaps`	Функция, которая возвращает способности телефонии по адресу коммуникационного устройства
`lineGetAddressStatus`	Функция, которая возвращает текущий статус коммуникационного устройства по его адресу
`lineGetAddressID`	Функция получает идентификационный номер ID коммуникационного устройства на основе адреса
Открытие и закрытие устройств линии	
`lineOpen`	Функция, которая открывает указанное устройство для обеспечения последующего мониторинга и/или контроля линии
`lineClose`	Функция, которая закрывает открытую линию

Таблица 2.1 *(продолжение)*

Функция	Описание
Функции работы с форматами адресов (номерами телефонов)	
lineTranslateAddress	Функция служит для конвертирования формата адреса (номера телефона) из канонического формата в формат dialable и обратно
lineSetCurrentLocation	Функция устанавливает текущее месторасположение
lineSetTollList	Данная функция управляет так называемым списком потерь
lineGetTranslateCaps	Функция возвращает адрес устройства линии на основе анализа его возможностей
Работа с событиями и состояниями	
lineGetCallInfo	Получение информации о вызове
lineGetCallStatus	Эта функция возвращает текущее состояние указанного вызова
lineSetAppSpecific	Установка собственных данных в поле dwAppSpecific информационной структуры вызова LINECALLINFO
Функции ассистента телефонии	
lineRegisterRequest Recipient	Эта функция регистрирует приложение как получателя входящих вызовов с указанным типом соединения. Поле dwRequestMode описывает тип принимаемых вызовов, созданных на отдаленной стороне, и принимает следующие LINEREQUESTMODE_ константы: LINEREQUESTMODE_DROP — соединение запрашивает функция tapiRequestDrop; LINEREQUESTMODE_MAKECALL — соединение запрашивает функция tapiRequestMakeCall; LINEREQUESTMODE_MEDIACALL — соединение запрашивает функция tapiRequestMediaCall
lineGetRequest	Эта функция запрашивает и возвращает следующий оговоренный с отдаленной стороной на указанном соединении режим. В зависимости от параметра dwRequestMode функцией в буфер памяти будут возвращены различные структуры данных. Например, если dwRequestMode — LINEREQUESTMODE_MAKECALL, функция вернет данные в структуре LINEREQMAKECALL. Если же dwRequestMode — LINEREQUESTMODE_MEDIACALL, функция вернет данные в структуре LINEREQMEDIACALL

Таблица 2.1 *(продолжение)*

Функция	Описание
Размещение исходящих вызовов	
`lineMakeCall`	Создание исходящего вызова
`lineDial`	Функция набирает номер и производит дозвон
Принятие входящих вызовов	
`lineAnswer`	Функция дает ответ на входящий вызов
Работа со звонками	
`lineSetNumRings`	Эта функция устанавливает количество звонков, которые должны быть получены для входящего вызова на данном устройстве линии, прежде чем устройство линии должно начать процедуру ответа на вызов
`lineGetNumRings`	Эта функция возвращает количество звонков, которые должны быть получены для входящего вызова на данном устройстве линии, прежде чем устройство линии должно начать процедуру ответа на вызов
Привилегии	
`lineSetCallPrivilege`	Эта функция устанавливает привилегию приложения TAPI
Закрытие и завершение вызова	
`lineDrop`	Устанавливает вызов в состояние "Idle" и разрывает соединение
`lineDeallocateCall`	Уничтожает вызов и освобождает дескриптор
Работа с дескрипторами вызовов	
`lineHandoff`	Данная функция передает право монопольного использования указанного вызова другому приложению. Приложение может быть или определено непосредственно по имени файла, или косвенно, как самое высокое приоритетное приложение, которое обрабатывает запросы указанного режима носителей
`lineGetNewCalls`	Данная функция возвращает дескрипторы приложений, обращающихся к указанному устройству линии. Приложение должно иметь привилегию мониторинга, чтобы использовать данную функцию
`lineGetConfRelatedCalls`	Эта функция возвращает список дескрипторов вызовов, которые являются частью конференции. Указанный вызов должен участвовать в конференц-связи

Таблица 2.1 *(окончание)*

Функция	Описание
Получение и установка информации о месторасположении абонента	
`lineTranslateDialog`	Эта функция выводит на экран диалоговое окно, которое позволяет пользователю изменять текущие настройки местоположения и параметры визитной карточки
`lineGetCountry`	Получение информации о стране. Данная информация может применяться, например, для установки правил набора номера

Вспомогательные функции TAPI

После разбора базовых функций TAPI мы кратко остановимся на расширенных функциях. Также стоит заметить, что во вспомогательном наборе функций присутствуют и функции из базового набора, но с расширенными возможностями, поэтому в табл. 2.2 некоторые функции будут дублироваться.

В табл. 2.2 представлен краткий список вспомогательных функций и структур TAPI, разбитый по категориям работы.

Таблица 2.2. *Вспомогательные функции TAPI*

Функция	Описание
Установка режима работы однонаправленного канала	
`lineSetCallParams`	Производит изменение в параметрах существующего вызова
Контроль носителей	
`lineMonitorMedia`	Включает или отключает прием соответствующих сообщений, посылаемых TAPI при изменении указанных режимов носителей на указанной линии
Контроль и сбор цифровых данных	
`lineMonitorDigits`	Включает или отключает прием соответствующего сообщения, посылаемого TAPI на получение цифровых данных
`lineGatherDigits`	Принимает цифровые данные, поступившие по вызову в буфер

Таблица 2.2 *(продолжение)*

Функция	Описание
Контроль тонового сигнала в линии	
`lineMonitorTones`	Настраивает ожидание и поиск тонового сигнала на указанном вызове
Управление режимами носителей (тип несущего потока в соединении)	
`lineSetMediaControl`	Устанавливает режим(ы) носителей на указанном вызове
`lineSetMediaMode`	Устанавливает режим(ы) носителей на указанном вызове в структуре `LINECALLINFO`
Генерация сигналов и тонов	
`lineGenerateDigits`	Посылает заданную последовательность цифр в линию, используя указанный способ передачи сигналов
`lineGenerateTone`	Посылает тоновый сигнал в линию
Прием и переназначение вызовов	
`lineAccept`	Сообщает другим приложениям, что данное приложение берет на себя ответственность за прием указанного вызова
`lineRedirect`	Переадресовывает указанный вызов другому приложению
Закрытие и завершение вызова	
`lineDrop`	Устанавливает вызов в состояние "Idle" и разрывает соединение
Постановка вызова на удержание	
`lineHold`	Ставит на удержание данный вызов
`lineUnHold`	Восстанавливает с удержания указанный вызов
Принятие входящих вызовов	
`lineSecureCall`	Делает существующий вызов "безопасным" в плане вмешательств в данный вызов других событий. Например, установки отложенного звонка
Передача вызова на другого абонента	
`lineSetupTransfer`	Подготавливает указанный вызов для передачи его другому абоненту

Таблица 2.2 *(продолжение)*

Функция	Описание
`lineCompleteTransfer`	Непосредственно передает вызов, установленный для передачи функцией `lineSetupTransfer`, другому абоненту
`lineBlindTransfer`	Передает вызов другой стороне
`lineSwapHold`	Меняет активный вызов на вызов, находящийся на удержании
Конференц-связь	
`lineSetupConference`	Настраивает вызов для подключения к другой стороне
`linePrepareAddTo Conference`	Подготавливает абонента для подключения к существующей конференц-связи
`lineAddToConference`	Добавляет вызов к существующей конференц-связи
`lineRemoveFromConference`	Отключает вызов от существующей конференц-связи
Перевод вызова на фиксированный номер телефона	
`linePark`	Закрепляет данный вызов на фиксированный номер телефона
`lineUnPark`	Восстанавливает закрепленный вызов
Посылка информации к отдаленной стороне	
`lineReleaseUserUserInfo`	Освобождает буфер хранения информации пользовательского режима и разрешает системе записывать новую информацию в данный буфер
`lineSendUserUserInfo`	Посылает информацию пользовательского режима на указанном вызове на отдаленную сторону
Завершение вызова	
`lineCompleteCall`	Завершает вызов на указанном соединении
`lineCompleteUnCall`	Отменяет завершение вызова на указанном соединении
`Call Pickup`	Дает возможность пользователям ответить на входящий вызов в других местах. Вы можете установить группы телефонных пользователей, которые могут отвечать на входящие вызовы. Когда один член группы не может ответить на вызов, любой из других членов может забрать данный вызов к себе. Абоненты обычно принадлежат только одной группе Call Pickup, которая является отделом или группой людей, имеющих подобные функции

Таблица 2.2 *(продолжение)*

Функция	Описание
linePickup	Забирает вызов, поступивший на другой номер. (Функция linePickup может также использоваться для режима отложенного звонка)
Терминал	
lineSetTerminal	Определяет терминал-устройство для указанного устройства линии, адреса или вызова
Приоритеты	
lineGetAppPriority	Возвращает handoff и/или получает информацию о приоритетах ассистента телефонии
lineSetAppPriority	Устанавливает handoff и/или заполняет информацию о приоритетах ассистента телефонии
Управление поставщиком услуг	
lineAddProvider	Устанавливает соответствующую службу телефонной связи поставщика услуг
lineConfigProvider	Открывает диалоговое окно настройки конфигурации указанного поставщика услуг
lineRemoveProvider	Удаляет существующую службу телефонной связи поставщика услуг
lineGetProviderList	Возвращает список инсталлированных служб телефонной связи
Агент автоматического распределения вызовов (ACD)	
lineAgentSpecific	Позволяет приложению обратиться к функциям агента автоматического распределения вызовов, связанного с указанным адресом
lineGetAgentActivityList	Получает список действий агента, из которого приложение может выбирать соответствующие функции, которые агент может выполнить
lineGetAgentCaps	Получает связанные с агентом возможности, которые поддерживаются на указанном устройстве линии
lineGetAgentGroupList	Получает список групп, в которых агент может регистрировать в качестве автоматического распределителя вызовов
lineGetAgentStatus	Получает состояние агента по связанному с ним адресу
lineSetAgentActivity	Получает состояние активности агента по адресу

Таблица 2.2 *(окончание)*

Функция	Описание
`lineSetAgentGroup`	Регистрирует агента в группе
`lineSetAgentState`	Устанавливает агента в определенное состояние по его адресу
Прокси	
`lineProxyMessage`	Используется зарегистрированным обработчиком вызова для генерации сообщений TAPI, связанных с его ролью. Например, обработчик агента устройства автоматического распределения вызовов может использовать эту функцию, чтобы генерировать LINE_AGENTSTATUS-сообщения, которые будут получены всеми приложениями, имеющими открытую указанную линию
`lineProxyResponse`	Завершает работу зарегистрированного обработчика вызовов. Например, данная функция завершает работу обработчика агента устройства автоматического распределения вызовов на сервере
Качество обслуживания (Quality of service)	
`lineSetCallQualityOf Service`	Изменяет текущие параметры качества обслуживания для существующего вызова
Разное	
`lineSetCallData`	Устанавливает поле `CallData` структуры `LINECALLINFO`
`lineSetCallTreatment`	Устанавливает, какие звуки пользователь слышит, когда вызов является оставшимся без ответа или находится в ожидании
`lineSetLineDevStatus`	Устанавливает состояние устройства линии

Обработка сообщений линии TAPI

В этом разделе мы расскажем об обработке сообщений, пришедших с устройств линии TAPI. Обработка сообщений TAPI очень похожа на обработку сообщений операционной системы Windows. Далее мы расскажем вам об основных сообщениях, касающихся TAPI. Для начала разберем функцию обработки сообщений `LineCallback`.

LineCallback — функция обработки сообщений линии

Сообщения позволяют прикладным программистам реагировать на различные изменения, произошедшие в работе, а также сообщать пользователю об изменениях статуса. Как и другие API, TAPI обеспечивает механизм получения сообщений на основе функции обратного отзыва. Данная функция обеспечивает прием и обработку сообщений TAPI, переданных операционной системой Windows. Функция определена в виде прототипа в TAPI.PAS следующим образом:

```
TLineCallback = procedure(hDevice, dwMessage, dwInstance,
                    dwParam1, dwParam2, dwParam3: Longint);
```

Параметры функции перечислены в табл. 2.3.

Таблица 2.3. *Параметры функции* `LineCallback`

Параметр	Описание
`hDevice`	Содержит дескриптор на устройство линии. Вы можете определить характер сообщения с данного устройства, используя параметр `dwMessage`
`dwMessage`	Идентификатор полученного сообщения
`dwInstance`	Данный параметр не используется TAPI
`dwParam1`	Первый параметр сообщения
`dwParam2`	Второй параметр сообщения
`dwParam3`	Третий параметр сообщения

Используя данную функцию, вы сможете создавать собственные алгоритмы обработки сообщений TAPI. Прежде всего, обратим ваше внимание на то, что от версии к версии TAPI набор сообщений увеличивается. Например, так объявлены некоторые идентификаторы сообщений в TAPI.PAS:

```
const
LINE_ADDRESSSTATE = 0;
LINE_CALLINFO   = 1;
LINE_CREATE    = 19; // TAPI v1.4
/* вырезано */
LINE_AGENTSPECIFIC = 21; // TAPI v2.0
LINE_AGENTSTATUS  = 22; // TAPI v2.0
/* вырезано */
```

```
LINE_APPNEWCALLHUB = 32; // TAPI v3.0
LINE_CALLHUBCLOSE = 33; // TAPI v3.0
LINE_DEVSPECIFICEX = 34; // TAPI v3.0
```

Сообщения линии TAPI

Для установки способа общения с TAPI вы используете "механизмы договоров". Обрабатывая сообщение TAPI, стоит иметь в виду, что некоторые из сообщений могут быть только в старших версиях TAPI, и поэтому на программиста ложится задача не только правильно обработать сообщения, но и обработать их с учетом версии TAPI.

Мы не станем подробно описывать каждое сообщение, благо в документации по TAPI это сделано за нас, а остановимся на кратком обзоре сообщений, разбив их по версиям.

Итак, основные сообщения линии TAPI версий 2.0 и ниже представлены в табл. 2.4.

Таблица 2.4. *TAPI-сообщения версий 2.0 и ниже*

Сообщение	Параметры сообщения	Описание

`LINE_ADDRESS STATE`	`dwDevice = hLine;`
	`dwCallbackInstance = Callback;`
	`dwParam1 = idAddress;`
	`dwParam2 = AddressState;`
	`dwParam3 = (DWORD) 0;`
		Посылается вашему приложению всякий раз, когда изменяется адрес статуса открытого устройства линии. Вы также можете управлять посылкой и чтением очереди, используя соответствующие функции: `lineGetStatusMessages` и `lineSetStatusMessages`

`LINE_CALLINFO`	`dwDevice = hCall;`
	`dwCallbackInstance = hCallback;`
	`dwParam1 = CallInfoState;`
	`dwParam2 = (DWORD) 0;`
	`dwParam3 = (DWORD) 0;`
		Будет посылаться приложению всякий раз, когда изменяется информация входящего вызова. Также вы можете вызвать функцию `lineGetCallInfo` для того, чтобы получить текущую информацию о вызове

`LINE_CALLSTATE`	`dwDevice = hCall;`
	`dwCallbackInstance = hCallback;`
	`dwParam1 = CallState;`
	`dwParam2 = CallStateDetail;`
	`dwParam3 = CallPrivilege;`
		Будет посылаться приложению при изменении состояния входящего вызова. Вы можете вызвать функцию `LineGetCallStatus` для получения детальной информации о текущем состоянии вызова

Таблица 2.4 *(продолжение)*

Сообщение	Параметры сообщения	Описание
LINE_ CLOSE	dwDevice = hLine; dwCallbackInstance = hCallback; dwParam1 = (DWORD) 0; dwParam2 = (DWORD) 0; dwParam3 = (DWORD) 0;	Будет посылаться приложению, когда указанное устройство линии принудительно закроется. Дескриптор устройства линии или любые дескрипторы входящих вызовов линии далее будут недействительны после получения данного сообщения
LINE_ CREATE	dwDevice = 0; dwCallbackInstance = 0; dwParam1 = idDevice; dwParam2 = 0; dwParam3 = 0;	Будет послано приложению, чтобы сообщить ему, что создано новое устройство линии
LINE_ DEVSPE CIFIC LINE_ DEVSPE CIFIC FEATURE	dwDevice = hLineOrCall; dwCallbackInstance = hCallback; dwParam1 = DeviceSpecific1; dwParam2 = DeviceSpecific2; dwParam3 = DeviceSpecific3;	Посылается для уведомления приложения о зависящих от устройств линии событиях, произошедших на устройстве линии, адресе или вызове. Значение сообщения и интерпретация параметров зависят от определенного устройства
LINE_ GATHER DIGITS	dwDevice = hCall; dwCallbackInstance = hCallback; dwParam1 = GatherTermination; dwParam2 = 0; dwParam3 = 0;	Посылается для уведомления приложения о том, что поток буферизировал данные и цифровой вызов закончен или отменен. Полученные в буфере данные можно исследовать приложением
LINE_ GENERATE	dwDevice = hCall; dwCallbackInstance = hCallback; dwParam1 = GenerateTermination; dwParam2 = 0; dwParam3 = 0;	Посылается для уведомления приложения, что текущая цифровая или тоновая генерация закончились

Таблица 2.4 *(продолжение)*

Сообщение	Параметры сообщения	Описание
LINE_ LINEDEV STATE	dwDevice = hLine; dwCallbackInstance = hCallback; dwParam1 = DeviceState; dwParam2 = DeviceStateDetail1; dwParam3 = DeviceStateDetail2	Будет посылаться приложению, когда состояние устройства линии будет изменено. Для считывания нового состояния устройства линии вы можете вызвать функцию lineGetLineDevStatus
LINE_ MONITOR DIGITS	dwDevice = hCall; dwCallbackInstance = hCallback; dwParam1 = Digit; dwParam2 = DigitMode; dwParam3 = 0;	Будет посылаться приложению всякий раз, когда устройство линии обнаружит на входе цифровые данные. Для управления процессом посылки данного сообщения служит функция lineMonitorDigits
LINE_ MONITOR MEDIA	dwDevice = hCall; dwCallbackInstance = hCallback; dwParam1 = MediaMode; dwParam2 = 0; dwParam3 = 0;	Будет посылаться приложению всякий раз, когда устройство линии обнаружит изменение режима носителей. Для управления процессом посылки данного сообщения служит функция lineMonitorMedia
LINE_ MONITOR TONE	dwDevice = hCall; dwCallbackInstance = hCallback; dwParam1 = dwAppSpecific; dwParam2 = 0; dwParam3 = 0;	Будет послано приложению при обнаружении тоновой несущей на заданном устройстве линии. Для управления процессом посылки данного сообщения служит функция lineMonitorTones
LINE_ REPLY	dwDevice = 0; dwCallbackInstance = hCallback; dwParam1 = idRequest; dwParam2 = Status; dwParam3 = 0;	Этим сообщением TAPI информирует приложение о завершении работы асинхронной функции. Сообщение возвращает также результат работы данной функции

Таблица 2.4 *(окончание)*

Сообщение	Параметры сообщения	Описание

LINE_REQUEST	dwDevice = 0;
	dwCallbackInstance = hRegistration;
	dwParam1 = RequestMode;
	dwParam2 = RequestModeDetail1;
	dwParam3 = RequestModeDetail2;
		Информирует приложение о получении новых вызовов на устройстве линии от другого приложения

Дополнительные сообщения линии TAPI версий 2.0 и ниже представлены в табл. 2.5.

Таблица 2.5. *Дополнительные TAPI-сообщения версий 2.0 и ниже*

Сообщение	Описание
LINE_AGENTSPECIFIC LINE_AGENTSTATUS	Посылаются приложению при изменении статуса агента автоматического распределения вызовов (ACD). Узнать статус можно также функцией lineGetAgentStatus
LINE_APPNEWCALL	Посылается TAPI в тот момент, когда будет создан новый Call-дескриптор
LINE_PROXYREQUEST	Будет послано приложению, чтобы сообщить, что устройство линии было удалено в системе. Следует отметить, что данное сообщение не используется при временном отключении устройств, например, при извлечении PCMCIA-устройств. Данное сообщение используется именно при необратимом удалении, например, при удалении из устройств в системе. В этом случае поставщик услуг более не может следить за данным устройством

Сообщения TAPI версий 2.2 и выше представлены в табл. 2.6.

Таблица 2.6. *TAPI-сообщения версий 2.2 и выше*

Сообщение	Описание
LINE_AGENTSESSIONSTATUS	Будет послано приложению, когда состояние сессии агента (устройства) автоматического распределения вызовов (ACD) изменится на обработчик агента, для которого приложение в настоящее время имеет открытую линию. Это сообщение может быть сгенерировано с помощью функции lineProxyMessage

Таблица 2.6 *(окончание)*

Сообщение	Описание
LINE_QUEUESTATUS	Будет послано приложению, когда состояние очереди устройства автоматического распределения вызовов изменится на обработчик агента, для которого приложение в настоящее время имеет открытую линию. Это сообщение может быть сгенерировано с помощью функции lineProxyMessage
LINE_AGENT STATUSEX	Будет послано приложению, когда состояние устройства автоматического распределения вызовов изменится на обработчик агента, для которого приложение в настоящее время имеет открытую линию. Это сообщение может быть сгенерировано с помощью функции lineProxyMessage
LINE_GROUPSTATUS	Будет послано приложению, когда состояние группы устройства автоматического распределения вызовов изменится на обработчик агента, для которого приложение в настоящее время имеет открытую линию. Это сообщение может быть сгенерировано с помощью функции lineProxyMessage
LINE_PROXYSTATUS	Будет послано, когда доступные proxy-серверы изменятся на устройстве линии
TAPI LINE_APPNEWCALLHUB	Будет послано приложению для уведомления, что создан новый концентратор вызовов (call hub)
TAPI LINE_CALLHUBCLOSE	Будет послано приложению для уведомления, что концентратор вызовов (call hub) закрыт
TAPI LINE_DEVSPECIFICEX	Будут послано, чтобы уведомить приложение о зависящих от устройств событиях, появившихся на линии, адресе или вызове

Посылаемое сообщение всегда содержит дескриптор объекта, о поведении которого сообщение уведомляет.

Объектом могут быть: телефонное устройство, устройство линии или вызов. Для того чтобы определить, какой объект послал данное сообщение, надо исследовать параметры сообщения. Параметры каждого сообщения можно узнать в файле справки TAPI.

Помимо дескриптора TAPI-сообщение содержит информацию о конкретном произошедшем изменении в состоянии объекта. Ваше приложение всегда может получить полную информацию о состоянии объекта, вызывая так называемые функции "получения состояния".

Порядок поступления сообщений для входящих и исходящих вызовов

После того как мы кратко ознакомились с сообщениями TAPI, немного поговорим о порядке поступления сообщений в приложение. Данный раздел очень важен в понимании работы TAPI, и знания о порядке сообщений пригодятся вам в следующих разделах, рассказывающих о работе с вызовами.

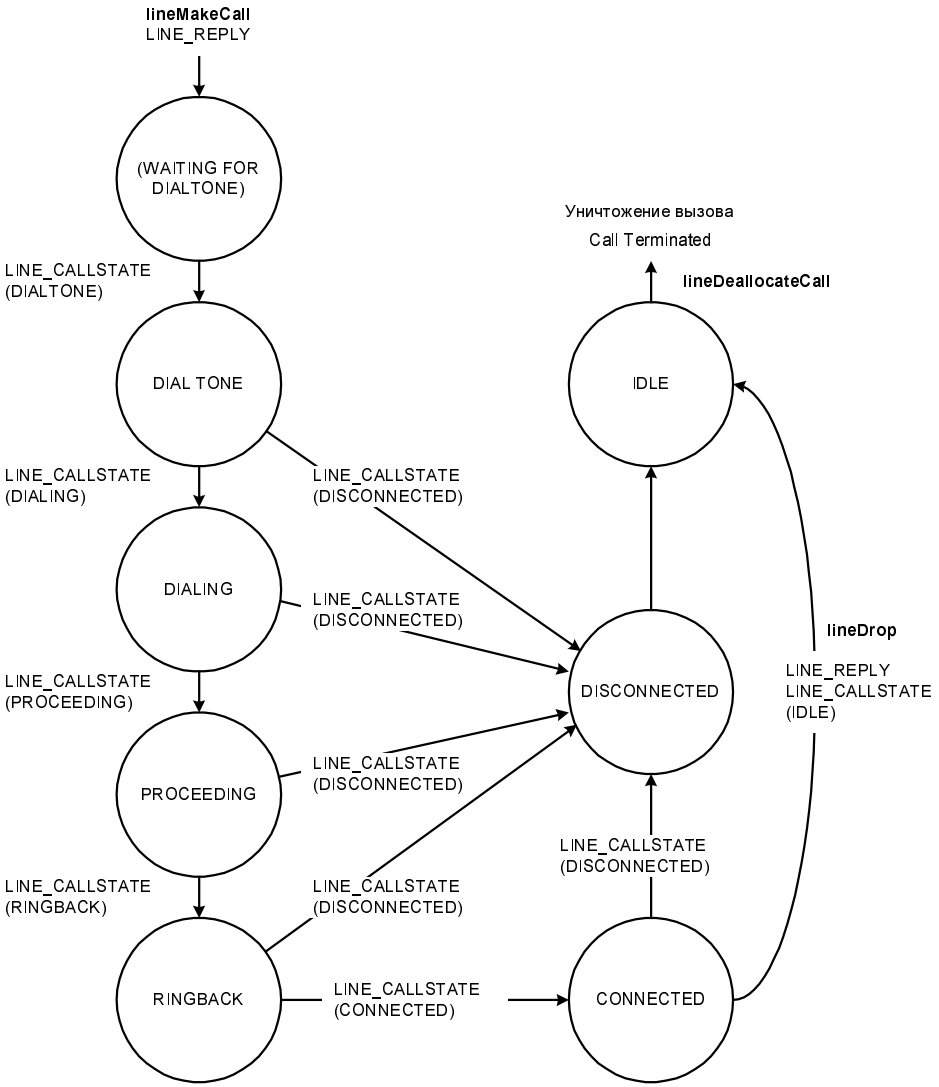


Рис. 2.4. Поступление сообщений TAPI в приложение, размещающее исходящий вызов

В этом разделе мы приведем всего две схемы, показывающие порядок прихода сообщений TAPI. На рис. 2.4 приведен порядок поступления сообщений в приложение TAPI, размещающее исходящий вызов, а на рис. 2.5 — приложение TAPI, принимающее входящий вызов. Стоит также заметить, что данные схемы отражают и порядок обработки соответствующих вызовов.

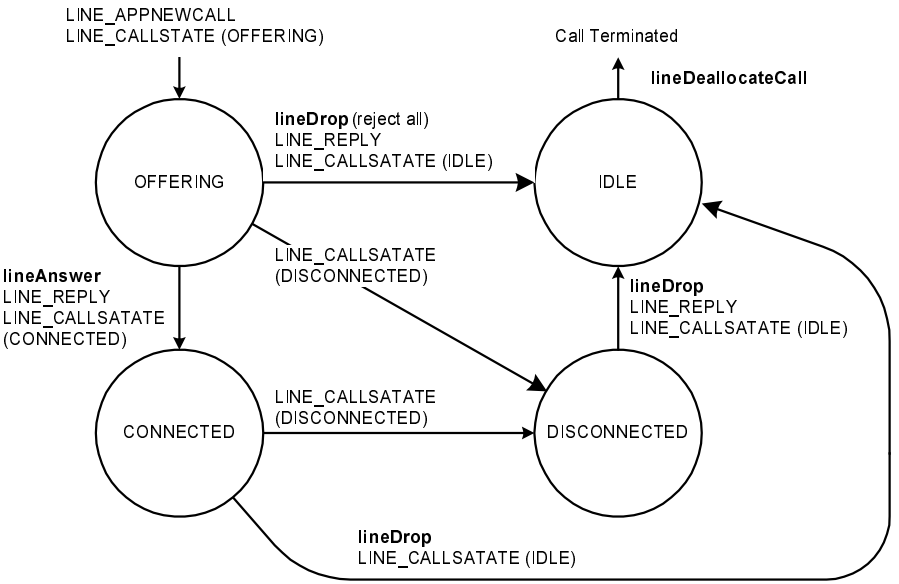


Рис. 2.5. Поступление сообщений TAPI в приложение, принимающее входящий вызов

Функции и структуры TAPI, связанные с обработкой сообщений

В TAPI есть несколько функций и структур, связанных с сообщениями. Кратко рассмотрим их.

Функция *lineGetMessage*

Эта функция возвращает сообщение телефонии, находящееся в очереди доставки для приложения, которое использует механизм уведомления Event Handle (получение уведомлений внутренним объектом приложения).

Функция имеет следующий синтаксис:

```
Function lineGetMessage(hLineApp: HLINEAPP;
    var lpMessage: TLineMessage; dwTimeout: DWORD): Longint;
                  stdcall; // TAPI v2.0
```

Параметры функции представлены в табл. 2.7.

Таблица 2.7. *Параметры функции* `lineGetMessage`

Параметр	Описание
`hLineApp`	Дескриптор линии, возвращенный `lineInitializeEx`. Приложение должно было установить константу `LINEINITIALIZEEXOPTION_USEEVENT` в поле `dwOptions` структуры `LINEINITIALIZEEXPARAMS`
`lpMessage`	Указатель на структуру `LINEMESSAGE`. При успешном выполнении функции данная структура будет содержать следующее сообщение, которое было поставлено в очередь
`dwTimeout`	Интервал времени ожидания (в миллисекундах). В течение данного времени функция ожидает сообщение. Если `dwTimeout` равна 0, то функция проверяет и возвращает поставленное в очередь сообщение немедленно. Если `dwTimeout` равна `INFINITE`, время ожидания функции нелимитировано, и функция ждет сообщение до тех пор, пока оно не придет

При успешном выполнении функция возвращает нуль, иначе возвращается код ошибки. Возможные коды ошибок при работе с данной функцией могут быть следующими: `LINEERR_INVALAPPHANDLE`, `LINEERR_OPERATIONFAILED`, `LINEERR_INVALPOINTER` и `LINEERR_NOMEM`.

Структура *LINEINITIALIZEEXPARAMS*

С данной структурой мы уже сталкивались при обсуждении инициализации TAPI. Как вы знаете, данная структура хранит дополнительные параметры инициализации.

Структура имеет следующий вид:

```
PLineInitializeExParams = ^TLineInitializeExParams;
lineinitializeexparams_tag = packed record
dwTotalSize, // TAPI v2.0
dwNeededSize, // TAPI v2.0
dwUsedSize, // TAPI v2.0
dwOptions: DWORD; // TAPI v2.0
Handles: TTAPIHandleUnion;
dwCompletionKey: DWORD; // TAPI v2.0
end;
TLineInitializeExParams = lineinitializeexparams_tag;
LINEINITIALIZEEXPARAMS = lineinitializeexparams_tag;
```

Поля структуры представлены в табл. 2.8.

Таблица 2.8. *Поля структуры* `LINEINITIALIZEEXPARAMS`

Поле	Описание
`dwTotalSize`	Полный размер структуры данных (байт)
`dwNeededSize`	Размер возвращенной информации (байт)
`dwUsedSize`	Размер той части структуры данных, которая содержит полезную информацию (байт)
`dwOptions`	Одна из констант, определяющих механизм уведомления *(см. разд. "Три механизма уведомлений (сообщений) TAPI")* Константы могут быть следующими: • `LINEINITIALIZEEXOPTION_CALLHUBTRACKING` • `LINEINITIALIZEEXOPTION_USECOMPLETIONPORT` • `LINEINITIALIZEEXOPTION_USEEVENT` • `LINEINITIALIZEEXOPTION_ USEHIDDENWINDOW`
`handles`	Если поле `dwOptions` имеет значение `LINEINITIALIZEEXOPTION_USEEVENT`, то TAPI возвращает в этом поле дескриптор
`dwCompletionKey`	Если поле `dwOptions` имеет значение `LINEINITIALIZEEXOPTION_USECOMPLETIONPORT`, то вы должны задать в этом поле дескриптор существующего порта завершения, созданного с использованием функции `CreateIoCompletionPort`. Если `dwOptions` имеет значение `LINEINITIALIZEEXOPTION_USECOMPLETIONPORT`, то вы должны определить в этом поле значение, которое было возвращено в параметре `lpCompletionKey` функции `GetQueuedCompletionStatus`, чтобы выделить сообщения завершения (completion message) как сообщения телефонии

Структура *LINEMESSAGE*

Структура `LINEMESSAGE` несет в себе данные, содержащие статусное состояние открытой приложением линии. Данная структура используется функцией `lineGetMessage` и имеет вид:

```
PLineMessage = ^TLineMessage;
linemessage_tag = packed record
hDevice, // TAPI v2.0
dwMessageID, // TAPI v2.0
dwCallbackInstance, // TAPI v2.0
dwParam1, // TAPI v2.0
dwParam2, // TAPI v2.0
dwParam3: DWORD; // TAPI v2.0
end;
```

```
TLineMessage = linemessage_tag;
LINEMESSAGE = linemessage_tag;
```

Поля структуры представлены в табл. 2.9.

Таблица 2.9. *Поля структуры* `LINEMESSAGE`

Поле	Описание
`hDevice`	Дескриптор устройства линии или вызова. Тип дескриптора (чей он) может быть определен на основании поля `dwMessageID`
`dwMessageID`	Устройство линии или вызов
`dwCallback Instance`	Данное поле TAPI не обрабатывается
`dwParam1; dwParam2; dwParam3`	Параметры сообщения

Функция *lineGetStatusMessages*

Данная функция запрашивает и получает информацию о состояниях устройства. На ее основе приложение может определить, какие сообщения будут посылаться от TAPI к приложению при изменениях состояний устройств в ходе работы.

Функция имеет следующий синтаксис:

```
function lineGetStatusMessages(hLine: HLINE;
         var dwLineStates,
           dwAddressStates: DWORD): Longint; stdcall;
```

Параметры функции представлены в табл. 2.10.

Таблица 2.10. *Параметры функции* `lineGetStatusMessages`

Параметр	Описание
`hLine`	Дескриптор устройства линии
`dwLine States`	Данный параметр представляет собой 32-битный массив флагов состояний устройства, сообщения об изменении которых должны быть посланы приложению. Если флаг установлен в `TRUE`, то сообщение будет получено приложением, иначе оно будет блокировано. Битовые флаги состояний описаны в документации в константах `LINEDEVSTATE_`

Таблица 2.10 *(окончание)*

Параметр	Описание
dwAddress States	Аналогично dwLineStates, за исключением того, что данный параметр управляет сообщениями, посылаемыми приложению при изменении адреса конкретного состояния устройства. Битовые флаги описаны в константах LINEADDRESSSTATE_

При успешном выполнении функция возвращает нуль, иначе возвращается код ошибки. Возможные коды ошибок при работе с данной функцией могут быть следующими: LINEERR_INVALLINEHANDLE,LINEERR_OPERATIONFAILED, INEERR_INVALPOINTER, LINEERR_RESOURCEUNAVAIL, LINEERR_NOMEM и LINEERR_UNINITIALIZED.

Функция *lineSetStatusMessages*

Данная функция устанавливает те состояния устройства, изменения которых в ходе работы спровоцируют TAPI на посылку соответствующего сообщения приложению. Например, состояние DISCONNECTED определяется константой LINEDEVSTATE_DISCONNECTED. При установке данного битового флага в TRUE приложение будет получать сообщение об обрыве связи, иначе сообщение будет блокировано. Данная функция является противоположной по значению функции lineGetStatusMessage и имеет следующий синтаксис:

```
function lineSetStatusMessages(hLine: HLINE;
 dwLineStates, dwAddressStates: DWORD): Longint; stdcall;
```

Параметры функции те же, что и функции lineGetStatusMessages.

Функция *lineSetCallPrivilege*

Эта функция устанавливает определенную привилегию.

Функция имеет следующий синтаксис:

```
function lineSetCallPrivilege(hCall: HCALL;
         dwCallPrivilege: DWORD): Longint; stdcall;
```

Параметры функции представлены в табл. 2.11.

Таблица 2.11. *Параметры функции* `lineSetCallPrivilege`

Параметр	Описание
hCall	Дескриптор (HCALL) на вызов, привилегия которого должна быть установлена

Таблица 2.11 *(окончание)*

Параметр	Описание
`dwCall Privilege`	Флаг привилегии. В данном параметре может быть установлена только одна привилегия из констант `LINECALLPRIVILEGE_`. Константа `LINECALLPRIVILEGE_MONITOR` указывает на то, что приложение может только контролировать вызов (режим мониторинга). Константа `LINECALLPRIVILEGE_OWNER` указывает на то, что приложение может манипулировать вызовом.

В этом разделе мы кратко рассмотрели сообщения TAPI и немного затронули вызовы, далее мы постараемся более подробно рассмотреть работу с вызовами в TAPI.

Размещение исходящих вызовов TAPI

В данном разделе мы разберем работу с исходящими вызовами TAPI. Под термином "исходящие вызовы" будем подразумевать подготовку устройства линии, набор номера и установление связи с удаленной стороной.

Исследуем два способа работы: один основан на использовании ассистента телефонии; другой — на низкоуровневых функциях TAPI. Оба этих способа используют номера (адреса) телефонов для установления соединения. Прежде чем приступить к разбору функций TAPI, служащих для работы с исходящими вызовами, остановимся на форматах номеров телефонов, представленных в TAPI.

Форматы номеров телефонов в TAPI

В TAPI формат номера, или по-другому адреса телефонов, представлен в двух вариантах: каноническом формате и формате dialable (дословно — формат, регламентирующий правила набора номера для устройства линии). Стоит также отметить, что существует еще так называемый визуальный адрес, который фактически является основным номером телефона, выводящимся на экран пользователя без специальных или управляющих символов.

Так как форматы наборов номера телефона в разных странах различны, существует международный стандарт, регламентирующий основные правила набора телефонных номеров, действующий во всех странах мира. Например, стандарт регламентирует правила набора номеров для международных телефонных соединений.

Данный формат в TAPI является каноническим и удобен для хранения в различных базах данных. Канонический адрес или номер телефона представляет собой строку ASCII, которая содержит некие символы в определенном порядке. Все канонические адреса начинаются с символа плюс (+). Этот символ имеет функцию идентификации строки телефонного номера как канонического адреса. Такая строка содержит в себе код страны (представленный переменной длиной цифр, завершающейся пробелом), затем следует дополнительный код города (переменная длина цифр, окруженных скобками) и в конце — номер абонента. Номер абонента состоит из цифр, которые могут быть представлены в dialable-формате.

Теперь мы обсудим формат dialable-адресов. Данный формат включает в себя, помимо цифр, служебные (управляющие) элементы, регламентирующие правила набора номера для устройства линии. В табл. 2.12 мы коротко остановимся на них.

Таблица 2.12. *Элементы dialable-адресов*

Элемент	Описание
Dialable Number	Набор: • от 0 до 9 • от A до D • (* # , ! W w P p T t @ $?) • ^ CRLF (#13#10)
!	Вставка паузы (половина секунды) в строку набора номера
P или p	Этот знак указывает, что устройство линии (модем) должно переключиться в импульсный режим набора номера
T или t	Этот знак указывает, что устройство линии (модем) должно переключиться в тоновый режим набора номера
,	Этот знак указывает на то, что устройство линии должно сделать паузу при наборе номера. Продолжительность паузы зависит от способностей устройства линии. Вы можете использовать многократные запятые, чтобы обеспечить более длинные паузы
W или w	Этот знак указывает на то, что устройство линии должно дождаться сигнала линии для продолжения набора. Часто этот знак используют при выходе на междугороднюю станцию
@	Этот знак указывает на то, что устройство линии должно дождаться так называемого "тихого ответа" и после его получения продолжить набор. "Тихий ответ" обычно представляет собой кратковременный гудок (тоновый сигнал), сопровождаемый несколькими секундами тишины

Таблица 2.12 *(окончание)*

Элемент	Описание
$	Этот знак указывает на то, что устройство линии должно дождаться специального сигнала "billing signal". Данный сигнал используется, например, для передачи информации с кредитных карт
?	Этот символ указывает, что устройство линии запросит у пользователя оставшийся номер
;	Этот символ, если он помещен в конце частично указанного dialable-адреса, указывает на то, что информация номера неполная и дополнительная номерная информация будет передана чуть позже. Использовать данный знак позволяется только в DialableNumber части адреса
|	Этот дополнительный символ указывает, что вся информация после этого знака и до знаков (+), (|), (^), (CRLF) должна быть обработана устройством линии как информация под-адреса. Под-адреса широко используются в ISDN
Под-адрес	Строка, содержащая под-адрес и заканчивающаяся знаками (+), (|), (^), (CRLF) или концом адресной строки
^	Этот дополнительный символ указывает на то, что вся информация после него до следующего (CRLF) или конца dialable-адреса должна быть обработана как имя цифровой сети комплексного обслуживания
Имя цифровой сети комплексного обслуживания	Строка, содержащая информацию об имени и оканчивающаяся (CRLF) или концом dialable-адреса
CRLF *(#13#10)*	Эта дополнительная символьная пара указывает на то, что dialable-номер следует за другим dialable-номером

Dialable-адрес очень важен, т. к. многие из функций TAPI применяют его. Используя Dialable-адреса, вы как бы управляете устройством линии. Например, можете переключать тональный или импульсный набор простым добавлением управляющего знака в номере телефона.

Ассистент телефонии

Ассистент телефонии (Assisted Telephony) является, наверное, самым простым по использованию сервисом TAPI. Ассистент телефонии помогает прикладному программисту с минимальными затратами создать приложение, реализующее функции дозвона, набора номера вызываемого абонента, а также обработку исходящих вызовов. Данный сервис TAPI может использовать-

ся, чтобы обеспечить простой способ набора номера телефона из ваших приложений.

Фактически ассистент телефонии работает следующим образом: при вызове соответствующих функций ассистент телефонии устанавливает связь с так называемым сервером телефонии, после чего сервер телефонии, используя TAPI, производит набор номера и устанавливает соединение. Функции ассистента телефонии начинаются с префикса tapi.

Функции ассистента телефонии

Ассистент телефонии представляет разработчику четыре функции: `tapiRequestMakeCall`, `tapiGetLocationInfo`, `tapiRequestMediaCall` и `tapiRequestDrop`. Так как последние две функции являются устаревшими и включены в TAPI для совместимости со старыми приложениями, мы не будем обсуждать их.

Функция *tapiRequestMakeCall*

Функция `tapiRequestMakeCall` будет пытаться установить обычное голосовое соединение между пользователем и удаленной стороной. Фактически данная функция просто производит набор номера, после чего при успешном соединении предоставляет пользователю возможность разговаривать по телефону.

Функция имеет следующий синтаксис:

```
function tapiRequestMakeCall(lpszDestAddress,
      lpszAppName, lpszCalledParty,
      lpszComment: LPCSTR): Longint; stdcall;
```

Параметры функции представлены в табл. 2.13.

Таблица 2.13. *Параметры функции* `tapiRequestMakeCall`

Параметр	Описание
`lpszDest Address`	Номер (адрес) телефона. Адрес может быть представлен как в каноническом формате, так и в формате dialable. Обратите внимание, что данный адрес не будет проверен на корректность набора. Также максимальная длина адреса ограничена `TAPIMAXDESTADDRESSSIZE`, включая нуль-терминатор строки
`lpszAppName`	Имя приложения, использовавшего данную функцию. Максимальная длина ограничена `TAPIMAXAPPNAMESIZE`, включая нуль-терминатор строки
`lpszCalled Party`	Информация о вызываемом абоненте. Максимальная длина ограничена `TAPIMAXCALLEDPARTYSIZE`, включая нуль-терминатор строки

Таблица 2.13 *(окончание)*

Параметр	Описание
lpszComment	Дополнительный комментарий. Максимальная длина комментария ограничена TAPIMAXCOMMENTSIZE, включая нуль-терминатор строки

Следующий пример показывает использование данной функции:

```
TAPIResult := TapiRequestMakeCall(
      PChar('242020'), // номер телефона
      PChar(Application.Title) // приложение
      '', '');
```

Функция *tapiGetLocationInfo*

Функция tapiGetLocationInfo возвращает код страны и код города (области), указанные в текущих настройках телефонии. Приложение может использовать эту информацию, чтобы помочь пользователю сформировать надлежащие канонические номера телефонов.

Функция имеет следующий синтаксис:

```
function tapiGetLocationInfo(lpszCountryCode,
             lpszCityCode: LPCSTR): Longint; stdcall;
```

Установление вызова с помощью низкоуровневых функций линии

Одной из основных задач телефонных приложений является работа с вызовами. Как было сказано ранее, под термином "вызовы" мы подразумеваем установку связи с удаленной стороной. Однако мы немного расширим формулировку, и под термином "вызовы" будем понимать любое (исходящее или, наоборот, входящее) соединение, устанавливающее связь межу двумя адресатами (абонентами).

Для работы с исходящими вызовами (соединениями), т. е. для подготовки устройства линии, набора номера и т. п., в TAPI существуют низкоуровневые функции устройства линий.

Как только ваше приложение открыло устройство линии, вы можете создать (или по-другому разместить) исходящий вызов, используя функцию lineMakeCall. В процессе размещения вызова вы определяете адрес вызова (номер телефона и код города) в параметре lpszDesAddress. Режим передачи данных задается в параметре lpCallParams.

В случае успеха данная функция возвратит положительный "идентификатор вызова" или отрицательный код ошибки.

Коды ошибок описывают определенные состояния. Так, например, код ошибки LINEERR_CALLUNAVAIL указывает на то, что линия находится в использовании (кто-то кроме вас уже установил вызов).

Если вызов номера заканчивается успешно, приложению будет послано определенное сообщение LINE_REPLY (асинхронный ответ на lineMakeCall), показывающее завершение вызова.

Однако получение данного сообщения не говорит еще об успешном установлении соединения, скорее, оно сообщает об успешном использовании функции lineMakeCall, а также указывает на то, что дескриптор, полученный данной функцией, является правильным. В данном случае имейте в виду, что вызов хоть и размещен, но он еще не является успешным или, попросту говоря, рабочим.

Перед установкой связи вызов может пройти несколько различных состояний. Смена каждого из этих состояний влечет посылку определенного сообщения приложению. Например, могут быть следующие состояния вызовов: набор номера, ожидание сигнала линии и т. д. Данные сообщения, уведомляющие приложение о смене этих состояний, отражены в группе сообщений LINE_CALLSTATE.

Фактически только при приеме сообщения LINECALLSTATE_CONNECTED приложение устанавливает успешное соединение и может начать посылать данные.

Что можно сказать о самих посылаемых данных? Помимо компьютерных данных (посылка файлов или других данных по телефонной линии) в TAPI возможно установить голосовой режим передачи данных. Программная модель TAPI позволяет, помимо передачи компьютерных данных, устанавливать передачу данных голосом.

Та же самая функция lineMakeCall используется для установки типа передаваемых данных. Если в параметре lpCallParams.dwBearerMode данной функции определена константа LINEBEARERMODE_DATA, вызов будет настроен для передачи компьютерных данных. Для установки другого режима передачи (передача голосом, передача факса) вы должны использовать другие значения. Если вы определите в данном поле ноль, то TAPI установит голосовое соединение на частоте 3,1 кГц. На этой частоте можно передавать речь, факс и медиаданные.

Однако мы снова должны подчеркнуть, что, используя функции низкого уровня для размещения вызовов (как исходящих, так и входящих), вы должны ориентироваться на способности устройства линии, определяемые приложением при инициализации. Это существенно, и об этом не нужно забывать.

Прежде чем вызывать функцию `lineMakeCall`, вы должны настроить параметры вызова, хранящиеся в структуре данных `LINECALLPARAMS`. Функция `lineMakeCall` имеет параметр `CallParams`, который указывает на данную структуру. Как мы упомянули, работая с данной структурой, вы определяете качество обслуживания, которое вы хотите запросить от сети. В ней вы также можете определить разнообразные параметры установки вызова ISDN.

Если вы не используете данную структуру, TAPI сам обеспечит установку настроек на голосовой вызов по умолчанию.

Если вы хотите, чтобы ваше приложение настроило специальные параметры установки вызова (например, при соединении по ISDN), вы должны правильно настроить данную структуру данных.

Специальная поддержка вызова номера

Набор номера является одним из основных функций телефонной связи. Поскольку функция `lineMakeCall` служит в основном для простых вызовов, то для более сложных в TAPI предусмотрена функция `lineDial`.

Рассмотрим, как работает данная функция.

Сначала вы должны установить вызов передачи или конференц-связи. После того как TAPI автоматически распределит вызов, вы вызываете `lineDial`, чтобы фактически запустить набор номера в вызове.

Если необходимо, вы можете вызвать `lineDial` несколько раз, обеспечивая многоступенчатый набор номера, при условии поддержки данной операции устройством линии. Список номеров для многоступенчатого набора передается в функцию в виде строки, где номера отделены символом CRLF (#13#10).

Конечно, различные поставщики услуг поддерживают различные функциональные возможности. Те поставщики услуг, которые поддерживают мультиплексирование при установке соединений, могут установить индивидуальный физический вызов с каждым из адресов.

> ***Мультиплексирование в двух словах***
>
> Принцип действия обычного мультиплексора прост: поступающие по нескольким входящим низкоскоростным линиям сигналы передаются в отведенном для каждого из них частотном диапазоне или интервале времени по высокоскоростной исходящей линии. На противоположном конце высокоскоростной линии эти сигналы вычленяются, или демультиплексируются. То есть реально на одном канале одновременно может быть установлено несколько соединений с разными адресатами.

Поставщики услуг, поддерживающие мультиплексирование, на каждое устанавливаемое соединение возвращают индивидуальный дескриптор вызова.

Как и прежде, Windows сообщит приложению все состояния вызова посредством сообщений LINE_CALLSTATE.

Если хотите предоставить пользователю возможность досрочно прерывать вызов, вы должны использовать функцию lineDrop.

Для того чтобы указать TAPI, что набор номера закончен, вы можете установить пустую строку в параметре lpszDestAddress функции lineDial. Однако сделать это можно только при условии, что в параметре lpszDestAddress в предыдущих вызовах функций lineMakeCall и lineDial были переданы номера (адреса) телефонов, разделенные точкой с запятой.

Функция lineDial, как и все остальные функции TAPI, может возвратить различные результаты своей работы. Например, если функция возвращает LINEERR_INVALADDRESS, то данная ошибка означает, что никакой набор номера не был произведен.

Далее мы более подробно остановимся на самих функциях и структурах.

Функция *lineDial*

Эта функция набирает dialable-номер на указанном вызове.

Функция имеет следующий синтаксис:

```
function lineDial(hCall: HCALL; lpszDestAddress: LPCSTR;
        dwCountryCode: DWORD): Longint; stdcall;
```

Параметры функции представлены в табл. 2.14.

Таблица 2.14. *Параметры функции* `lineDial`

Параметр	Описание
hCall	Дескриптор исходящего вызова, где должен быть набран номер в формате dialed. Приложение должно быть владельцем вызова. Состояние вызова *hCall* может быть любым, кроме "простоя" и "разъединения"
lpszDest Address	Номер в формате dialed
dwCountry Code	Код страны адресата. Данное поле используется TAPI для выбора протокола передачи вызова. Если значение данного поля установлено 0, то служба поставщика услуг установит протокол, заданный по умолчанию

При успешном завершении функция возвращает идентификатор вызова или отрицательный код ошибки. Данная функция, помимо этого, посылает код

ошибки и в параметре dwParam1 сообщения LINE_REPLY или посылает 0, если функция завершена успешно.

Функция *lineMakeCall*

Эта функция размещает вызов на устройстве линии.

```
function lineMakeCall(hLine: HLINE; lphCall:
          PHCall; lpszDestAddress: LPCSTR;
          dwCountryCode: DWORD;
          CallParams: PLineCallParams): Longint; stdcall;
```

Параметры функции представлены в табл. 2.15.

Таблица 2.15. *Параметры функции* `lineMakeCall`

Параметр	Описание
hLine	Дескриптор устройства линии
lphCall	Дескриптор исходящего вызова. Дескриптор будет действителен только после получения сообщения TAPI — LINE_REPLY. Используйте данный дескриптор при работе с другими функциями TAPI, работающими с вызовами. Данный параметр будет недействителен, если функция возвратит ошибку
lpszDest Address	Номер в формате dialed. Данный номер может быть пустой строкой, если вы устанавливаете "горячее" соединение или соединение с помощью LineDial. В последнем случае lineMakeCall размещает вызов, т. к. если бы он находился в состояние "Dial Tone". Использование данной функции при работе с поставщиками услуг, которые поддерживают мультиплексирование, позволяет приложению задавать многократные адреса
dwCountry Code	Код страны адресата. Данное поле используется TAPI для выбора протокола передачи вызова. Если значение данного поля установлено 0, то служба поставщика услуг установит протокол, заданный по умолчанию
CallParams	Указатель на структуру LINECALLPARAMS. Эта структура позволяет приложению необходимую настройку вызова

При успешном завершении функция возвращает идентификатор вызова или отрицательный код ошибки. Данная функция, помимо этого, посылает код ошибки и в параметре dwParam2 сообщения LINE_REPLY или посылает 0, — если функция завершена успешно.

Структура *LINECALLPARAMS*

Структура LINECALLPARAMS описывает параметры, устанавливаемые при создании вызовов. Данную структуру используют функции lineMakeCall и

TSPI_lineMakeCall, а также она применяется как параметр и в других действиях. Например, ее использует lineOpen функции.

Если ваше приложение требует устройство-специфичные расширения (device-specific extensions), вы должны использовать следующие поля: dwDevSpecificSize и dwDevSpecificOffset, задающие размер и смещение выделенной для хранения области расширенных данных.

Обратите также внимание, что DialParams через dwDevSpecificOffset будут проигнорированы, когда структура передана в lineOpen функции.

Поля dwPredictiveAutoTransferStates через dwCallingPartyIDOffset будут доступны только тем приложениям, которые открывают устройство линии с версией TAPI 2.0 или выше. Поле dwAddressType будет доступно только приложениям, использующим версию TAPI 3.0 или выше.

Структура имеет очень много параметров, поэтому приводить здесь вид структуры мы не будем, а перейдем сразу к описанию полей. Итак, поля структуры LINECALLPARAMS, представлены в табл. 2.16.

Таблица 2.16. *Поля структуры* LINECALLPARAMS

Поле	Описание
dwTotalSize	Это поле указывает в байтах полный размер выделенной области памяти переменного размера, хранящей данную структуру данных. Этот размер должен быть достаточно большим, чтобы область памяти могла вместить все фиксированные и переменные части структуры
dwBearerMode	Это поле указывает режим работы канала вызова. Данное поле использует одну из констант LINEBEARERMODE_. Если dwBearerMode является нулевым, значение по умолчанию устанавливается равным LINEBEARERMODE_VOICE
dwMinRate	Это поле указывает минимальное значение скорости передачи данных, которое требуется для потока данных вызова в битах в секунду. При создании вызова поставщик услуг сделает попытку обеспечить самую высокую доступную скорость в требуемом диапазоне
dwMaxRate	Это поле указывает максимальное значение скорости передачи данных, которое требуется для потока данных вызова в битах в секунду
dwMediaMode	Это поле указывает вид вызова или, попросту говоря, характеризует тип потока данных, передаваемых в данном вызове (голос, факс и др.). Данное поле использует одну из констант LINEMEDIAMODE_. Если dwMediaMode является нулевым, значение по умолчанию становится LINEMEDIAMODE_INTERACTIVEVOICE

Таблица 2.16 *(продолжение)*

Поле	Описание
dwCallParam Flags	Это поле содержит в себе коллекцию булевых флагов, устанавливающих параметры вызова. Данное поле использует константы LINECALLPARAMFLAGS_
dwAddressMode	Это поле указывает режим определения адреса. Данное поле использует одну из констант LINEADDRESSMODE_. Значение поля не может быть установлено в режим LINEADDRESSMODE_ADDRESSID для функции lineOpen
dwAddressID	Это поле хранит адресный идентификатор, если dwAddressMode установлен в LINEADDRESSMODE_ADDRESSID
DialParams	Это поле указывает параметры набора номера LINEDIALPARAMS, которые используются на этом вызове. Если значение поля определено равным нулю, устанавливаются значения по умолчанию, заданные в поле DefaultDialParams структуры LINEDEVCAPS
dwOrig AddressSize	Это поле указывает (в байтах) размер области памяти переменного размера, содержащей структуру данных для оригинального адреса. Формат этого адреса зависит от поля dwAddressMode
dwOrig AddressOffset	Это поле указывает (в байтах) смещение области памяти переменного размера, содержащей структуру данных для оригинального адреса от начала структуры LINECALLPARAMS. Формат этого адреса зависит от поля dwAddressMode
dwDisplayable AddressSize	Это поле указывает размер строки, используемый для хранения визуализированного формата адреса (адреса, показываемого пользователю на экране без спецсимволов). Значения данного поля используются в сообщении LINECALLINFO
dwDisplayable AddressOffset	Это поле указывает смещение, заданное от начала структуры LINECALLPARAMS строки, используемой для хранения визуализированного формата адреса. Значения данного поля используются в сообщении LINECALLINFO
dwCalled PartySize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей информацию о вызываемом абоненте. Эта информация может быть определена приложением, которое делает вызов. Формат области памяти переменного размера должен быть установлен в dwStringFormat, как в LINEDEVCAPS
dwCalled PartyOffset	Это поле указывает смещение, заданное от начала структуры LINECALLPARAMS области памяти переменного размера, содержащей информацию о вызываемом абоненте. Формат области памяти переменного размера должен быть установлен в dwStringFormat, как в LINEDEVCAPS

Таблица 2.16 *(продолжение)*

Поле	Описание
dwCommentSize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей комментарии. Эта информация может быть определена приложением, которое делает вызов. Формат области памяти переменного размера должен быть установлен в dwStringFormat, как в LINEDEVCAPS
dwCommentOffset	Это поле указывает смещение, заданное от начала структуры LINECALLPARAMS области памяти переменного размера, содержащей комментарии. Формат области памяти переменного размера должен быть установлен в dwStringFormat, как в LINEDEVCAPS
dwUserUser InfoSize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей информацию о пользовательском режиме
dwUserUser InfoOffset	Соответствующее смещение от начала структуры LINECALLPARAMS. Поле дискриминатора протокола для режима пользовательской информации должно быть указано в первом байте данных, на которые указывает dwUserUserInfoOffset, и должно быть включено в dwUserUserInfoSize
dwHighLevel CompSize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей информацию о совместимости высокого уровня
dwHighLevel CompOffset	Соответствующее смещение от начала структуры LINECALLPARAMS
dwLowLevel CompSize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей информацию о совместимости нижнего уровня
dwLowLevel CompOffset	Соответствующее смещение от начала структуры LINECALLPARAMS
dwDevSpecific Size	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей устройство-специфичные расширения (device-specific extensions) или, по-другому, устройство-специфичную информацию
dwDevSpecific Offset	Это поле указывает смещение, заданное от начала структуры LINECALLPARAMS области памяти переменного размера, содержащей устройство-специфичные расширения (device-specific extensions) или, по-другому, устройство-специфичную информацию
dwPredictive AutoTransfer States	Данное поле содержит константы LINECALLSTATE_, определяющие состояние вызова

Таблица 2.16 *(продолжение)*

Поле	Описание
dwTarget AddressSize	Это поле указывает размер (в байтах) строки, содержащей dialable-адрес получателя (не dwAddressID), используемый TAPI автоматически в некоторых случаях
dwTarget AddressOffset	Соответствующее смещение от начала структуры LINECALLPARAMS
dwSending FlowspecSize	Данное поле содержит полный размер (в байтах) структуры FLOWSPEC, определенной в WinSock2 для QOS (Quality Of Service). Данная структура определяет качество обслуживания, желательное при установке исходящего соединения. Определенная поставщиком услуг часть данных после структуры FLOWSPEC не должна содержать указатели на другие блоки памяти, поскольку TAPI не сможет собрать данные, ссылки на которых заданы в указателях, и передать эти данные через взаимодействие процессов к приложению
dwSending FlowspecOffset	Соответствующее смещение от начала структуры LINECALLPARAMS
dwReceiving FlowspecSize	Данное поле содержит полный размер (в байтах) структуры FLOWSPEC, определенной в WinSock2 для QOS (Quality Of Service). Данная структура определяет качество обслуживания, желательное при установке входящего (принимающего) соединения. Определенная поставщиком услуг часть данных после структуры FLOWSPEC не должна содержать указатели на другие блоки памяти, поскольку TAPI не сможет собрать данные, ссылки на которые заданы в указателях, и передать эти данные через взаимодействие процессов к приложению
dwReceiving FlowspecOffset	Соответствующее смещение от начала структуры LINECALLPARAMS
dwDevice ClassSize	Это поле указывает размер (в байтах) строки (заканчивающейся NULL-символом), содержащей класс устройства, конфигурация которого определена в DeviceConfig. Допустимые строки класса устройства те же, что определены для функции lineGetID
dwDevice ClassOffset	Соответствующее смещение от начала структуры LINECALLPARAMS
dwDevice ConfigSize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей структуру VarString, которая содержит в себе настройки и параметры конфигурирования коммуникационного устройства. Если размер поля равен нулю, устройство линии использует настройки по умолчанию
dwDevice ConfigOffset	Соответствующее смещение от начала структуры LINECALLPARAMS

Таблица 2.16 *(окончание)*

Поле	Описание
dwCallDataSize	Это поле указывает размер (в байтах) прикладных данных, первоначально присоединенных к вызову
dwCallData Offset	Соответствующее смещение от начала структуры LINECALLPARAMS
dwNoAnswer Timeout	Данное поле содержит интервал повтора набора номера при дозвоне (в секундах) в состояниях PROCEEDING или RINGBACK. TAPI будет через заданное время делать попытку соединения, пока не будет автоматически остановлен поставщиком услуг, установившим состояния устройства линии в LINECALLSTATE_DISCONNECTED и LINEDISCONNECTMODE_NOANSWER. Если в данном поле указан ноль, то это означает, что приложение не желает автоматического повтора набора номера
dwCalling PartyIDSize	Размер строки идентификатора вызывающего абонента (в байтах), включая нулевой оконечный знак
dwCalling PartyIDOffset	Соответствующее смещение от начала структуры LINECALLPARAMS к строке с нулевым символом в конце, которая определяет идентификатор стороны, помещающей вызов. Если содержание идентификатора приемлемо и путь доступен, поставщик услуг передает идентификатор к вызываемому абоненту. Размер поля определен dwCallingPartyIDSize
dwAddressType	Данное поле содержит адресный тип, используемый для вызова. Это поле доступно при использовании версии TAPI 3.0 или выше

Константы *LINECALLPARAMFLAGS*

Данные константы определяют различные флаги вызова и представлены в табл. 2.17.

Таблица 2.17. *Константы LINECALLPARAMFLAGS_*

Константа	Описание
LINECALLPARAM FLAGS_BLOCKID	Эта константа указывает на то, что идентификатор отправителя вызова должен быть скрытный (Block caller ID)
LINECALLPARAM FLAGS_ DESTOFFHOOK	Этот константа указывает на то, что телефон вызываемой стороны должен быть автоматически взят

Таблица 2.17 *(продолжение)*

Константа	Описание
LINECALLPARAM FLAGS_IDLE	Эта константа указывает на то, что вызов должен начаться на незанятом соединении. Если при использовании функции lineMakeCall константа LINECALLPARAMFLAGS_IDLE не будет установлена и на устройстве линии есть существующий вызов, то функция прервет существующий вызов и начнет новый. Если нет никакого существующего вызова, функция начнет делать новый вызов
LINECALLPARAM FLAGS_NOHOLD CONFERENCE	Эта константа используется только вместе с функциями TAPI lineSetupConference и linePrepareAddToConference. Адрес, который будет добавлен к конференции с текущего вызова, определен в поле TargetAddress структуры LINECALLPARAMS. Данный вызов физически не будет получать сигнал "ответ станции" от коммутатора, но состояния его будут проходить через различные стадии установки соединения (например, "dialing" (набор номера)). Когда вызов достигает состояния "connected", это будет означать, что конференция автоматически установлена. Оригинальный (первоначальный) вызов, который остался в состоянии "connected", переключится в состояние "conferenced" и hConfCall установится в состояние "connected". Если вызов потерпит неудачу (например, произойдет разъединение, вызванное простоем), hConfCall введет состояние "Idle" ("не занято"), и первоначальный вызов останется в состоянии "connected"
LINECALLPARAM FLAGS_ONE STEPTRANSFER	Эта константа используется только вместе с функцией lineSetupTransfer и объединяет действия функции lineSetupTransfer и функции набора номера lineDial в отдельный шаг. Адрес, который будет набран по номеру, определен в поле TargetAddress структуры LINECALLPARAMS. Первоначальный вызов будет помещен в состояние "onholdpendingtranfer", как будто функцию lineSetupTransfer вызывали обычным способом, и вызов был установлен как обычно
LINECALLPARAM FLAGS_ ORIGOFFHOOK	Этот константа указывает на то, что телефон отправителя должен быть автоматически взят
LINECALLPARAM FLAGS_ PREDICTIVEDIAL	Эта константа используется только при размещении вызова на устройстве линии, поддерживающей способность телефонии, — LINEADDRCAPFLAGS_PREDICTIVEDIALER (узнать, поддерживает или нет устройство линии данную способность, можно, анализируя поле dwAddrCapFlags структуры LINEADDRESSCAPS)

Таблица 2.17 *(окончание)*

Константа	Описание
LINECALLPARAM FLAGS_SECURE	Эта константа указывает на то, что вызов должен быть безопасным

Функция *lineTranslateAddress*

Эта функция переводит указанный формат адреса на другой формат. Функция имеет следующий синтаксис:

```
function lineTranslateAddress(hLineApp: HLINEAPP;
    dwDeviceID, dwAPIVersion: DWORD;
    lpszAddressIn: LPCSTR; dwCard,
    dwTranslateOptions: DWORD;
    lpTranslateOutput: PLineTranslateOutput): Longint; stdcall;
```

Параметры функции представлены в табл. 2.18.

Таблица 2.18. *Параметры функции* `lineTranslateAddress`

Параметр	Описание
hLineApp	Дескриптор, возвращенный lineInitializeEx
dwDeviceID	Идентификатор устройства линии
dwAPIVersion	Версия TAPI
lpszAddressIn	Строка адреса-источника. Формат данного адреса должен быть каноническим или dialable. Данный параметр не должен быть Nil. Если lpszAddressIn содержит под-адрес, поле имени или дополнительные адреса, отделенные от первого адреса знаками ASCII CR и LF (перевод строки), функция переведет только первый адрес, остальные адреса будут возвращены в остававшейся части строки без модификации
dwCard	Идентификатор кредитной карточки, которая используется для набора номера. Это поле допустимо только при условии того, что установлен CARDOVERRIDE бит в dwTranslateOptions. Данный параметр определяет постоянный идентификатор кредитной карты, заданный в секции [Cards] системного реестра. Данный идентификатор может быть получен функцией lineTranslateCaps

Таблица 2.18 *(окончание)*

Параметр	Описание
dwTranslate Options	Формат перевода адреса (номера). Данный параметр задает последовательность связанных операций, которые будут выполнены до трансляции адреса в dialable-строку. Этот параметр может использовать следующие константы LINETRANSLATEOPTION_: • LINETRANSLATEOPTION_CARDOVERRIDE • LINETRANSLATEOPTION_CANCELCALLWAITING • LINETRANSLATEOPTION_FORCELOCAL • LINETRANSLATEOPTION_FORCELD
lpTranslate Output	Указатель (PLineTranslateOutput) на область памяти, содержащей результат работы функции. Результат перевода будет возвращен в структуре LINETRANSLATEOUTPUT. Прежде чем вызвать функцию lineTranslateAddress, вы должны установить поле dwTotalSize равным размеру структуры, чтобы TAPI выделил правильное количество памяти для размещения информации

Структура *LINETRANSLATEOUTPUT*

Структура LINETRANSLATEOUTPUT содержит результат перевода адреса. Данная структура имеет следующий вид:

```
PLineTranslateOutput = ^TLineTranslateOutput;
linetranslateoutput_tag = packed record
dwTotalSize,
dwNeededSize,
dwUsedSize,
dwDialableStringSize,
dwDialableStringOffset,
dwDisplayableStringSize,
dwDisplayableStringOffset,
dwCurrentCountry,
dwDestCountry,
dwTranslateResults: DWORD;
end;
TLineTranslateOutput = linetranslateoutput_tag;
LINETRANSLATEOUTPUT = linetranslateoutput_tag;
```

Поля структуры представлены в табл. 2.19.

Таблица 2.19. *Поля структуры* `LINETRANSLATEOUTPUT`

Поле	Описание
`dwTotalSize`	Указывает полный размер (в байтах) выделенной области памяти переменного размера, хранящей данную структуру данных
`dwNeededSize`	Размер (в байтах) возвращенной информации
`dwUsedSize`	Размер (в байтах) той части структуры данных, которая содержит полезную информацию
`dwDialable StringSize`	Содержит полный размер (в байтах) строки, заканчивающейся нулевым символом. Данная строка содержит оттранслированный формат адреса, который можно использовать с функциями `lineMakeCall`, `lineDial` и др., использующими dialable-формат адреса
`dwDialable StringOffset`	Смещение от начала структуры
`dwDisplayable StringSize`	Содержит полный размер (в байтах) строки, заканчивающейся нулевым символом. Данная строка содержит визуализируемый формат адреса, который можно выводить на экран пользователю
`dwDisplayable StringOffset`	Смещение от начала структуры
`dwCurrent Country`	Содержит код страны, установленный в настройках системы (CurrentLocation)
`dwDestCountry`	Содержит код страны вызываемого абонента. Значение данного поля можно передавать в параметре `dwCountryCode` функции `lineMakeCall`, а также других функций TAPI, использующих код страны вызываемого абонента (получателя). Если поле имеет нулевое значение, это означает, что адрес был передан `lineTranslateAddress` не в каноническом формате
`dwTranslate Results`	Содержит служебную информацию, переданную функцией после завершения процесса перевода формата. Данная информация может быть проанализирована приложением, например, для настройки элементов интерфейса пользователя. Поле использует следующие константы `LINETRANSLATERESULT_`: • `LINETRANSLATERESULT_CANONICAL` • `LINETRANSLATERESULT_INTERNATIONAL` • `LINETRANSLATERESULT_LONGDISTANCE` • `LINETRANSLATERESULT_LOCAL` • `LINETRANSLATERESULT_INTOLLLIST` • `LINETRANSLATERESULT_NOTINTOLLLIST`

Таблица 2.19 *(окончание)*

Поле	Описание
	• LINETRANSLATERESULT_DIALBILLING • LINETRANSLATERESULT_DIALQUIET • LINETRANSLATERESULT_DIALDIALTONE • LINETRANSLATERESULT_DIALPROMPT

Функция *lineTranslateDialog*

Эта функция выводит на экран диалоговое окно, которое позволяет пользователю изменять текущие настройки местоположения и параметры визитной карточки. Измененные в данном диалоговом окне настройки будут влиять только на то соединение, которое должно быть установлено (набрано по номеру). Функция имеет следующий синтаксис:

```
function lineTranslateDialog(hLineApp: HLINEAPP;
    dwDeviceID, dwAPIVersion: DWORD;
    hwndOwner: HWND; lpszAddressIn: LPCSTR): Longint;
                    stdcall; // TAPI v1.4
```

Параметры функции представлены в табл. 2.20.

Таблица 2.20. *Параметры функции* `lineTranslateDialog`

Параметр	Описание
`hLineApp`	Дескриптор, возвращенный `lineInitializeEx`
`dwDeviceID`	Идентификатор устройства линии
`dwAPIVersion`	Версия TAPI
`hwndOwner`	Хендл окна приложения
`lpszAddressIn`	Указатель на строку с нулевым символом в конце, содержащей номер телефона, который будет использоваться в диалоговом окне

Для демонстрации работы с исходящими вызовами мы создали небольшую программу, которая размещает вызов на первом устройстве линии (по умолчанию это обычно модем), производит дозвон на выбранный номер и устанавливает соединение. Внешний вид программы вы можете увидеть на рис. 2.6. В листинге 2.2 сокращенно представлена процедура размещения вы-

зова на устройстве линии. Полный исходный код программы вы сможете найти на прилагаемом компакт-диске в каталоге Source\Ch02\Ex02.

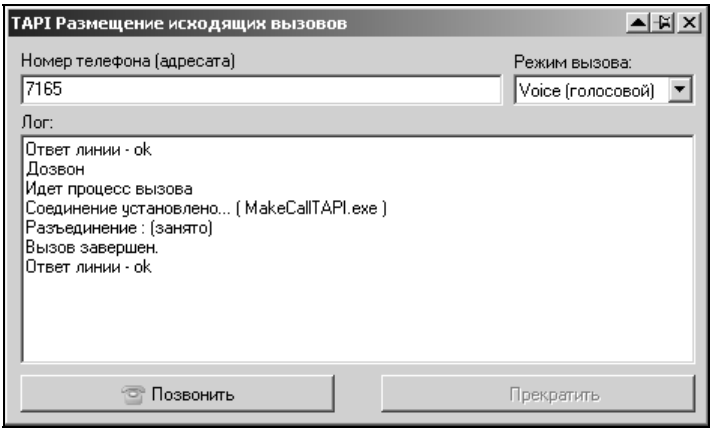


Рис. 2.6. Размещение исходящего вызова

Листинг 2.2. Размещение исходящего вызова на устройстве линии

```
procedure TFormMain.BitBtnCallingClick(Sender: TObject);
var
   BuffPhone : array[0..30] of char;
begin
   if Length(EditPhone.Text) > 0 then
   begin
      // < * * *  ВЫРЕЗАНО * * * >
      StrPCopy(BuffPhone, EditPhone.Text);
      // Режим носителей
      if ComboBox.ItemIndex = 0 then
         // ГОЛОСОВОЙ
         CallParams.dwMediaMode   := LINEMEDIAMODE_INTERACTIVEVOICE
      else
         CallParams.dwMediaMode := LINEMEDIAMODE_DATAMODEM;

      // Размещение исходящего вызова
      if lineMakeCall(line, call, BuffPhone, 0, @CallParams) < 0 then
         MemoLog.Lines.Add('Ошибка в lineMakeCall')
      else
      begin
          // < * * *  ВЫРЕЗАНО * * * >
      end;
   end
```

```
   else
      ShowMessage('Не выбран номер');
end;
```

Также на основе данного примера мы создали приложение мониторинга устройства линии. Стадии работы данного приложения вы можете увидеть на рис. 2.7. Исходный код данного примера находится в каталоге Source\Ch02\Ex03. В листинге 2.3 представлен сокращенный код процедуры обработки сообщений TAPI при работе с исходящим вызовом.

Листинг 2.3. Обработка сообщений TAPI при работе с исходящим вызовом

```
// Обработка сообщений TAPI
procedure lineCallback(hDevice, dwMsg, dwCallbackInstance,
                       dwParam1, dwParam2, dwParam3: LongInt);
{$IFDEF WIN32}
   stdcall;
{$ELSE}
   export;
{$ENDIF}
var
  Str  : string;
  hCall: THCall; // Дескриптор вызова
begin
  with FormMain do
  begin
    // Ответ ЛИНИИ
    if dwMsg = LINE_REPLY then
    begin
       // Если произошла ошибка - то код ошибки - отрицательное число
      // < * * *  ВЫРЕЗАНО * * * >
    end
    else
       // ГРУППА СООБЩЕНИЙ LINE_CALLSTATE
       if dwMsg = LINE_CALLSTATE then      { Изменено что-то на линии }
       begin
          hCall := THCall(hDevice);       // Дескриптор
          case dwParam1 of
            // Вызов в режиме ожидания или прекращен
            LINECALLSTATE_IDLE:
            Begin
                // Проверим дескриптор
                IF hcall <> 0 then
```

```
                begin
                   // Уничтожаем вызов и освобождаем память
                   lineDeallocateCall(hCall);

                end;
             end;
             // Приложение установило успешное соединение
             LINECALLSTATE_CONNECTED:
             Begin
                 // < * * *  ВЫРЕЗАНО * * * >
             end;
             // Идет процесс вызова
             LINECALLSTATE_PROCEEDING:
                       MemoLog.Lines.Add('Идет процесс вызова');
            // Дозвон
            LINECALLSTATE_DIALING   :
                       MemoLog.Lines.Add('Дозвон');
            // Разъединение
            LINECALLSTATE_DISCONNECTED:
            begin
                Str := 'Разъединение : ';
                // Причина ?
                if dwParam2 = LINEDISCONNECTMODE_NORMAL then
                   Str := Str + '(штатное)'
                else
                   if dwParam2 = LINEDISCONNECTMODE_BUSY then
                Str := Str + '(занято)';
                MemoLog.Lines.Add(Str);
             end;
            // Занято
            LINECALLSTATE_BUSY      : MemoLog.Lines.Add('Занято');
          end;
       end;
  end;    // with
end;
```

Стоит заметить, что эти два примера могут работать с одним и тем же устройством линии (модемом) одновременно. Это достигается за счет правильной установки привилегий.

Итак, кратко резюмируем данный раздел. В нем мы исследовали различные средства размещения исходящих вызовов. Мы также обсудили различные форматы адресов или номеров телефона, с которым работает TAPI. Однако для более полного ознакомления с TAPI мы также рассмотрим механизмы принятия входящих вызовов TAPI.

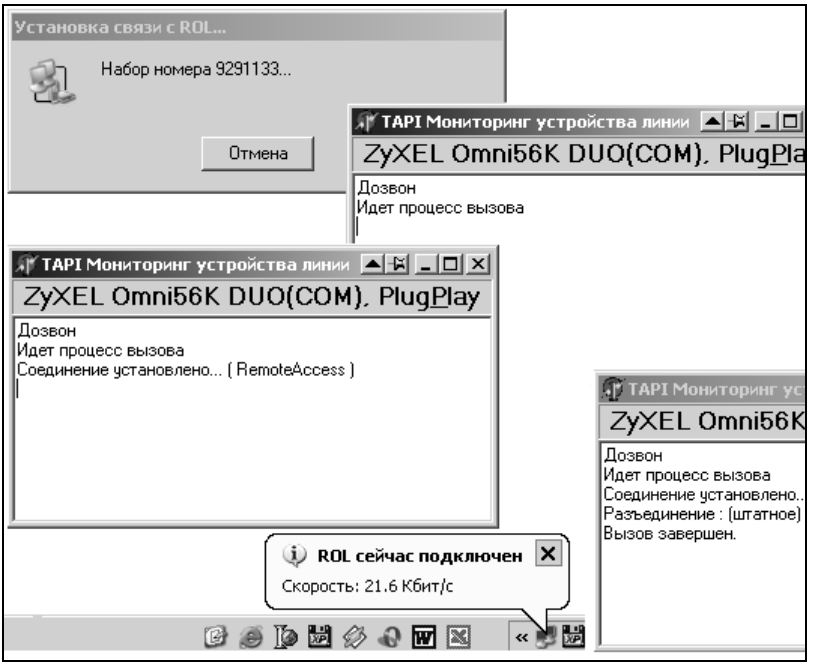


Рис. 2.7. Мониторинг устройства линии

Принятие входящих вызовов

В данном разделе мы постараемся рассказать вам о методике принятия входящих вызовов в TAPI. Разберем, какие шаги должно предпринять приложение TAPI, чтобы корректно обработать входящий вызов. Остановимся на функциях, которые используются для поддержки этих действий.

Поиск заинтересованного приложения

Когда на устройство линии приходит входящий вызов, в этот момент мы не знаем, какой вид вызова пришел. Это может быть обычный звонок по телефону, сделанный человеком, а может быть звонок, сделанный модемом. И в том и в другом случае мы еще не знаем, какой поток данных будет передаваться в соединении (голос, факс, данные). Здесь встает небольшая проблема, каким TAPI-приложением обрабатывать данный вызов, ведь одновременно в памяти могут висеть: программа по приему факсов, автоответчик или программа передачи данных. Какому из этих приложений передать для работы вызов. Как TAPI определить, что именно данная программа должна сделать: обработать входящий вызов, включить автоответчик для голосового соедине-

ния или запустить прием факсов для факсового соединения, а не сделать все наоборот.

Данная проблема решается просто. Как вы знаете, TAPI использует различные режимы, идентифицирующие виды вызовов. Данные режимы мы будем называть режимами носителей, т. е. режимами, задающими тип несущего потока в соединении. Они имеют большое значение в работе TAPI. На основе режимов, задаваемых приложением, TAPI определяет, какому из приложений передать вызов.

Когда вызов прибывает, информация о вызове, в частности информация о режиме носителей, анализируется TAPI (анализ происходит на основе считывания режима из поля `dwMediaMode` структуры `LINECALLINFO`).

Как только TAPI определит вид вызова, им сразу же будет сделана попытка найти заинтересованное в вызове приложение, зарегистрированное в TAPI.

Следуя нехитрым правилам, основанным на текущем состоянии системы и анализе информации, взятой из системного реестра, можно сказать, что TAPI делает следующие шаги для обработки входящих вызовов.

1. Поставщик услуг уведомляет динамически компонуемую библиотеку TAPI о получении входящего вызова.
2. TAPI исследует информацию, находящуюся в разделе системного реестра [`HandoffPriorities`], для получения списка TAPI-приложений, которые интересуются данным видом вызова (если таковые имеются).
3. Программный интерфейс для телефонии определяет первое перечисленное приложение как самое высокое приоритетное приложение. Если это приложение выполняется в настоящее время и имеет открытую линию для требуемого потока данных, установленных в режиме носителей, то устанавливается монопольное соединение (вызов переключается на обработку данным приложением в монопольном режиме). Если же в данный момент времени приложение не запущено и нет открытых приложением линий, то это приложение пропускается. TAPI снова использует информацию в системном реестре, чтобы найти заинтересованное приложение, которому и отдаст пришедший вызов.
4. Если же ни одно из приложений, перечисленных в системном реестре, не находится в надлежащем состоянии, то TAPI ищет другие выполняющиеся в данный момент TAPI-приложения, имеющие открытую линию для заданного режима носителей (хотя они и не перечислены в системном реестре). Относительный приоритет выбора этих неперечисленных приложений является произвольным и не связан с последовательностью запуска приложений или последовательностью операций открытия линии.

Если ваше приложение имеет специфическую линию, открытую для мониторинга, то данное приложение также получит дескриптор ко всем вызовам на этой линии.

Если же никакое приложение не станет владельцем вызова, TAPI, в конечном счете, отклонит вызов. Однако это случится именно тогда, когда не будет найдено заинтересованное приложение, которое сможет стать владельцем запроса.

Неизвестный режим носителей

Когда приходит входящий запрос, программный интерфейс для телефонии постоянно контролирует его продвижение. Сначала TAPI проверит первое сообщение `LINE_CALLSTATE`, переданное поставщиком услуг, и определит режим носителя прибывшего запроса. Если режим носителя определен, то TAPI передает управление заинтересованному приложению.

Однако возникают случаи, когда режим носителя не обозначен явно, т. е. в поле `dwMedia` структуры `LINECALLINFO` выставлен соответствующий бит (флаг) — `unknown`. В данном случае TAPI пошлет сообщение `LINEMEDIAMODE_UNKNOWN` и запустит так называемый процесс исследования.

Сначала TAPI определит, есть ли приложения, готовые принять вызов с неизвестным режимом носителя (далее для краткости — НВН-приложения). Их может быть несколько.

Если есть НВН-приложения, готовые принять вызов, работа TAPI пойдет следующим путем:

1. Прежде всего, TAPI проверит, запущено ли такое приложение и открыта ли им соответствующая линия.
2. Далее TAPI.DLL даст в монопольное использование самому высокому (в плане приоритета) НВН-приложению соответствующий дескриптор входящего вызова. Такое НВН-приложение примет сообщение `LINE_CALLSTATE` с параметром `dwParam3`, где TAPI передаст ему владельца.
3. Стоит также заметить, что TAPI передаст соответствующий дескриптор вызова и другим приложениям, открытым на данной линии, но только тем из них, которые находятся в режиме мониторинга.
4. Затем НВН-приложение может пытаться само определить режим носителя или задействовать помощь других TAPI-приложений. В последнем случае другие приложения исполнили бы собственное исследование данного режима носителя.
5. В первом же случае НВН-приложение может (без помощи стороннего TAPI-приложения) просто передать вызов определенному TAPI-приложению, вызвав функцию `lineHandoff`, т. е. в данном случае НВН-при-

ложение должно своими силами провести исследование поля dwMediaMode структуры LINECALLINFO и определить TAPI-приложение, заинтересованное в данном режиме носителя.

Если же нет TAPI-приложений, заинтересованных в приеме вызова с неизвестным режимом носителя (или данное приложение не открыло линию), далее TAPI пойдет следующим путем:

1. TAPI.DLL примет на себя обязанность НВН-приложения и начнет анализ всех TAPI-приложений для поиска наиболее подходящего приложения. Такое приложение TAPI определяет следующим образом. Как вы знаете, приложения при инициализации и открытии линии устанавливают вид обрабатываемого режима носителя, так вот на основе этих данных (поле dwMedia структуры LINECALLINFO) TAPI и попытается определить приложение, имеющее самый высокий приоритет режима носителей. То есть TAPI пробежится по всем приложениям и выберет приложение с наибольшим установленным приоритетом режима носителей.
2. Если же такое приложение не будет найдено, но на линии есть приложения, осуществляющие мониторинг, то для таких приложений вызов останется в состоянии "offering". Данным приложениям также TAPI передаст соответствующий дескриптор вызова. Если же вызов будет прекращен вызываемым абонентом, то в данном случае состояние вызова примет значение "Idle" (простой) и будет оставаться в нем до тех пор, пока приложения мониторинга не освободят дескриптор вызова.

Если же на линии нет приложений мониторинга, то TAPI просто отклонит вызов.

Приоритет режимов носителей

Кратко исследовав механизм приема вызовов, мы остановимся на том, что режимы носителей имеют следующие приоритеты.

1. LINEMEDIAMODE_INTERACTIVEVOICE
2. LINEMEDIAMODE_DATAMODEM
3. LINEMEDIAMODE_G3FAX
4. LINEMEDIAMODE_TDD
5. LINEMEDIAMODE_G4FAX
6. LINEMEDIAMODE_DIGITALDATA
7. LINEMEDIAMODE_TELETEX
8. LINEMEDIAMODE_VIDEOTEX
9. LINEMEDIAMODE_TELEX

10. LINEMEDIAMODE_MIXED
11. LINEMEDIAMODE_ADSI

Обязанности приложения, принимающего входящие вызовы

Приложение, принимающее входящие вызовы, имеет некоторые обязанности, накладываемые TAPI. Наиболее важная обязанность приложения — исследование и установка соответствующего режима носителя вызова.

Если исследование режима носителя прошло успешно, то приложение должно установить соответствующий бит, характеризующий режим. Если же исследование закончилось неудачей, то приложение должно очистить неудавшийся режим носителя, обнулив соответствующий бит в структуре LINECALLINFO. Далее приложение должно передать вызов следующему приложению (вызов передается приложению, имеющему самый высокий приоритет), которое может далее продолжить исследование вызова. Если, в конце концов, никакие биты режима носителей не установлены, "handoff" даст сбой, т. к. подходящих владельцев для данного вызова не было найдено.

> ***Примечание***
>
> Термином "handoff" в TAPI обозначается передача обслуживания (например, передача вызова от одного приложения к другому).

Однако то, что в теории хорошо, на практике оборачивается некоторыми проблемами. Например, режим носителей вызова успешно идентифицирован приложениями TAPI, но в поле dwMediaMode структуры LINECALLINFO все еще выставлен бит unknown. Такая ситуация вводит неопределенность в действия приложения. Приложение, получившее данный входящий вызов, не может быть абсолютно уверено, что оно является самым высоким по приоритету приложением, связанным с данным режимом носителей.

Для устранения такой ситуации приложение должно гарантировать, что данный вызов идет именно самому высокому приоритетному приложению.

Для этого приложение должно предпринять следующие шаги:

1. Приложение должно вызвать функцию lineSetMediaMode, которая сбросит unknown-бит и установит идентифицированный бит режима носителей.
2. Далее приложение должно вызвать функцию lineHandoff, чтобы возвратить вызов обратно в TAPI, который примет задачу обнаружения самого высокого приоритетного приложения для этого режима носителей.
3. Если же данное приложение и является самым высоким приоритетным приложением, заданным для этого режима носителей, то данное приложение получит по окончании работы функции lineHandoff значение

LINEERR_TARGETSELF. Данное значение как бы скажет приложению: "Ты и есть самое высокое приоритетное приложение для того режима носителей, так что давай, работай с этим вызовом".

Приложение не теряет управление над вызовом, и данная особенность позволяет обращаться с вызовом, как обычно.

Если же запрос к lineHandoff покажет, что данное приложение не является самым высоким приоритетным приложением для идентифицированного режима носителей, то приложение должно освободить дескриптор вызова или переключиться в режим мониторинга (переключение должно осуществляться с позволения самого высокого приоритетного приложения, обрабатывающего данный вызов).

Подведем итоги

Пока unknown-бит остается установленным, принимающее вызов приложение находится в состоянии неопределенности и не знает, является ли оно самым приоритетным для данного режима носителей. Поэтому оно должно провести исследование и определить приложение, имеющее самый высокий приоритет для данного запроса.

Приложение может быть уверено, что оно является самым высшим по приоритету приложением только тогда, когда unknown-бит выключен.

Программный интерфейс для телефонии рекомендует, чтобы все приложения, заинтересованные в приеме вызовов с неизвестным режимом носителя, использовали заданные по умолчанию приоритеты при исследовании других TAPI-приложений, также готовых принять такие вызовы.

Использование такого метода приведет к тому, что вы, например, не услышите факс в голосовом приложении или голос в факсовой программе.

Если ваше приложение будет исследовать режимы носителей, то TAPI также рекомендует включать в такое приложение контроль носителей. Данный контроль осуществляется функцией lineMonitorMedia, которая определит поток (сигнал) и вернет режим носителя. Использования контроля поможет вам избежать так называемой потери вызова.

Регламент работы приложения, определяющего режим носителей

Когда заинтересованное приложение для данных режимов носителей будет выбрано, и TAPI передаст этому приложению вызов на обработку, то последнее должно выполнить определенные регламентные работы.

Если приложение приняло вызов, переданный ему из другого приложения, то данное приложение должно сначала проверить битовые флаги, установленные в поле dwMediaMode структуры LINECALLINFO.

Далее если приложение определило, что установлен только единственный бит (флаг) режима носителей, то приложение имеет дело именно с тем вызовом, с которым оно должно работать.

Как было сказано, если приложение определило, что в данном виде вызова установлен флаг `unknown`, то приложение должно запустить механизм исследования для поиска приложения, имеющего самый высокий приоритет режима носителя.

Если вдруг приложение определило, что в поле `dwMediaMode` структуры `LINECALLINFO` установлен более чем один бит, и на вызов не был дан ответ, то приложение должно вызвать функцию `lineAnswer`, чтобы продолжить исследование вызова.

С другой стороны, если на вызов уже был дан ответ, приложение может продолжить исследование. В данном случае нет необходимости приложению отвечать на приходящий вызов.

Если исследование выполнится успешно, то приложение должно заполнить поле `dwMediaMode` структуры `LINECALLINFO` специфическим режимом носителей, распознанным исследованием. Если по факту режим носителя — ожидаемый режим, то приложение может начать обрабатывать данный вызов. Иначе (если это другой режим носителей) приложение должно сначала пытаться передать вызов далее по цепочке для обнаружения приложения, готового обработать данный вызов с такими параметрами носителя.

Если же во время исследования произойдет сбой (например, другое приложение очистило флаги для этого режима носителя или передается вызов НВН-приложению), то в таком случает приложение должно освободить вызов, обработать его или возвратить вызов.

Если попытка передать вызов к НВН-приложению закончилась неудачей или НВН-приложение не запущено, тогда приложение, которое в настоящее время имеет вызов, может передать его другому приложению, имеющему самый высокий приоритет для режима носителей (в этом случае бит `unknown` остается включенным, для того чтобы процесс исследования смог продолжиться).

Если возникли сбои при передаче вызова (handoff), приложение должно выключить бит носителя, установить следующий приоритетный бит и сделать другую попытку handoff. Данная операция может продолжаться до тех пор, пока handoff успешно не завершится, или до той поры, пока все биты режимов не будут выключены (кроме бита `unknown`). В этом случае конечное обращение к `lineHandoff` потерпит неудачу, т. к. приложение фактически осталось единственным владельцем вызова. Данный отказ сообщит приложению, что оно должно отклонить вызов и затем освободить его дескриптор.

Принятие входящего вызова

После того как мы исследовали процесс работы с режимами носителей, можно приступить к обсуждению процесса принятия вызова. Для того чтобы ваше приложение принимало вызовы, оно должно зарегистрировать привилегии на вызов при открытии линии функцией `lineOpen`.

Если ваше приложение открыло устройство линии с привилегией `LINECALLPRIVILEGE_MONITOR` (мониторинг), то данное приложение будет получать сообщение `LINE_CALLSTATE` для каждого вызова, прибывающего в линию.

Если же приложение открыло устройство линии с привилегией `LINECALLPRIVILEGE_OWNER`, то оно будет получать сообщение `LINE_CALLSTATE` только в том случае, если данное приложение станет владельцем вызова или если ему будет перенаправлен вызов (handoff).

В случае handoff, TAPI даст приложению дескриптор входящего вызова. Приложение будет хранить дескриптор до тех пор, пока не освободится вызов.

Использование механизма установки привилегий позволяет TAPI корректно рассылать приложениям сообщение `LINE_CALLSTATE`, возникающее при прибытии вызова. А т. к. данное сообщение содержит в себе дескриптор поступившего вызова, привилегию и состояние вызова, то приложение всегда может получить заблаговременную информацию о вызове с помощью функции `lineGetCallInfo`. Фактически приложение получает информацию о вызове до ответа на него.

Вызов функции `lineGetCallInfo` при поступлении сообщения модифицирует информацию в структуре данных `LINECALLINFO`. Анализируя вызов и используя другую информацию, ваше приложение сможет определить, действительно ли оно должно ответить на вызов.

В частности, структура данных `LINECALLINFO` включают следующую информацию:

- режимы соединения, скорость передачи данных (бит/сек), данные о цифровом вызове;
- текущий режим носителей. Если режим определен как `Unknown`, то вызов идет с неизвестным форматом носителя. В таком случае могут быть установлены другие биты, указывающие возможный режим носителей на вызове;
- данные, указывающие на то, откуда произошел или передался вызов;
- внутренняя информация, описывающая вызов (идентификационный номер вызова, имя или номер стороны, пославшей вызов, и т. д.).

Сообщение LINE_CALLSTATE также уведомляет приложение о способе установки вызова (вызов установлен другим приложением или вручную пользователем). Однако получение данного сообщения приложением еще не означает, что устройство линии может сразу принять вызов. Прежде чем начать прием вызова, устройство линии должно привести себя в готовность. Приведенное в готовность устройство линии посылает сообщение LINE_LINEDEVSTATE, информирующее приложение о состоянии устройства линии. В средах ISDN-телефонии приложение, принимающее вызовы (с lineAccept), должно предварительно проверить битовый флаг LINEADDRCAPFLAGS_ACCEPTTOALERT.

Стоит также заметить, что в очень редких случаях в некоторых средах телефонии информация о вызове первоначально может не передаться приложению или передастся только одному из приложений, висящих на линии. То есть остальные приложения не смогут сразу получить информацию о вызове. Например, идентификатор вызывающей программы может не передаться в первоначальном звонке, в этом случае идентификатор будет неизвестен, однако как только TAPI его определит, приложению снова пошлется сообщение LINE_CALLINFO, уведомляющее о том, что идентификатор наконец-то определен в вызове.

Теперь мы можем обсудить специфику принятия вызова на различных типах сетей.

В обычной телефонной сети приложению для принятия вызова нужно использовать функцию lineAccept. Эта функция сообщает другим приложениям, что данное приложение берет на себя ответственность за прием входящего вызова. Функция lineAccept будет подробно рассмотрена в *разд. "Функции и структуры TAPI, управляющие приемом вызовов"*.

В сетях ISDN-процедура принятия вызова должна информировать пользователя о том, что приложение готово принять данный входящий вызов. Информирование пользователя осуществляется появлением соответствующего диалогового окна на компьютере. Если в вызове установлен флаг LINEADDRCAPFLAGS_ACCEPTTOALERT, то приложение должно обязательно вызвать функцию lineAccept.

Если приложение будет не в состоянии сразу же вызвать функцию lineAccept (приложение может быть занято обработкой другого вызова и т. д.), то по истечении времени ожидания сеть предположит, что станция данного вызова выключена или разъединена. В зависимости от этого вызов может быть отклонен, повторен (если стоит режим повторного набора), или вызывающей стороне будет послано сообщение "disconnect".

Для непосредственного ответа на вызов нужно воспользоваться функцией lineAnswer.

Важно!

Стоит заметить, что принятие вызова — это не то же самое, что ответ на вызов. В обычной телефонной сети ответ на вызов означает offhook. В сетях ISDN ответ на вызов означает, что коммутатор устанавливает вызов в состояние "connected". До ответа на вызов нет никакого физического подключения коммутатора к адресату, хотя вызов и стоит в цепочке от вызывающей программы до коммутатора. То есть вызов дошел до коммутатора, но еще не дошел до абонента. Фактически ответ на вызов происходит тогда, когда коммутатор переключает вызов на абонента, и абонент принимает его.

Кратко цепочка прохождения вызова такая:

1. Вызов приходит TAPI, далее TAPI пересылает вызов заинтересованному приложению.
2. Если приложение готово принять вызов, оно вызывает функцию `lineAccept`. Фактически оно бронирует данный вызов, т. к. в это время приложением могут обрабатываться другие вызовы.
3. Как только приложение готово ответить на вызов, оно вызывает функцию `lineAnswer`.

Завершение вызова

Размещая и принимая вызовы, программист не должен забывать завершать их. После того как вызов завершится, ваше приложение примет сообщение `LINE_CALLSTATE`, информирующее приложение, что состояние устройства линии изменилось. Состояние "disconnect" будет означать, что фактически произошел разрыв соединения. В этом случае вы можете сразу же отключиться от линии, используя функцию `lineDrop`, прежде чем вы получите сообщение "remotedisconnect". В предыдущих разделах мы обсуждали вопросы, связанные с открытием и закрытием устройства линии. Важно не забывать предпринимать определенные шаги, чтобы корректно завершить вызов и закрыть устройство линии. Порядок таких шагов можно описать так:

1. Приложение сначала должно вызвать функцию `lineDrop`, которая установит вызов в состояние "Idle". Такой вызов все еще будет существовать для приложений, обслуживающих его дескриптор (например, приложений-мониторов).
2. Если вызов должен быть окончательно удален, тогда нужно воспользоваться функцией `lineDeallocateCall`, которая должна освободить дескриптор. После этого вызов больше не будет существовать.
3. Если приложение больше не должно работать с данным устройством линии или, более того, не должно принимать от него вызовы, следует вызвать функцию `lineClose`. Ну и если приложение вообще хочет завершить работу с сеансом TAPI, тогда нужно не забыть вызвать `lineShutdown`.

Для закрепления материала мы создали небольшую программу, позволяющую осуществлять прием входящих вызовов. Нами не ставилась задача полностью выполнить все требуемые шаги, описанные в предыдущих разделах. Мы остановились на упрощенной схеме приема входящих вызовов, чтобы упростить задачу и сделать ее более доступной для понимания.

Внешний вид программы и стадии ее работы представлен на рис. 2.7. Полный исходный код программы вы сможете найти на прилагаемом компакт-диске в каталоге Source\Ch02\Ex04.

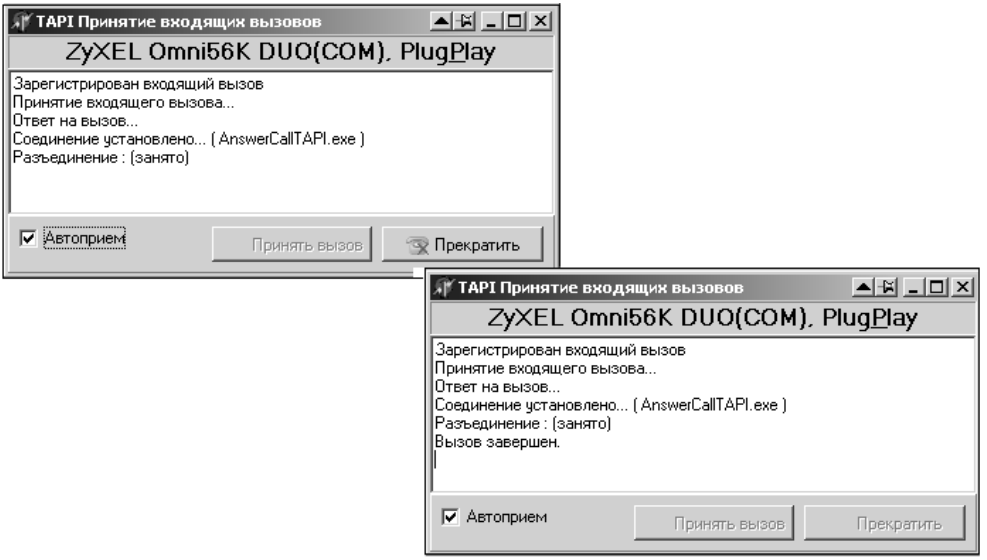


Рис. 2.8. Принятие входящего вызова

Сокращенный код процедуры обработки сообщений TAPI, поступающих в программу при приеме вызова, представлен в листинге 2.4.

Листинг 2.4. Пример процедуры обработки сообщений TAPI при приеме вызова

```
// Обработка сообщений TAPI
procedure lineCallback(hDevice, dwMsg, dwCallbackInstance,
                       dwParam1, dwParam2, dwParam3: LongInt);
{$IFDEF WIN32}
   stdcall;
{$ELSE}
   export;
{$ENDIF}
```

```
var
  Str  : string;
begin
  with FormMain do
  begin
       hCall := THCall(hDevice);       // Дескриптор
       // ГРУППА СООБЩЕНИЙ LINE_CALLSTATE
       if dwMsg = LINE_CALLSTATE then      { Изменено что-то на линии }
       begin
          case dwParam1 of
            // Переназначение
            LINECALLSTATE_OFFERING :
            Begin
                  MemoLog.Lines.Add('Зарегистрирован входящий вызов');
                  BitBtnAnswer.Enabled := True;
                  IF CheckBoxAuto.Checked then
                     BitBtnAnswerClick(nil);
            end;
            // Вызов в режиме ожидания или прекращен
            LINECALLSTATE_IDLE :
            Begin
                  // Проверим дескриптор
                  IF hcall <> 0 then
                  begin
                     // Уничтожаем вызов и освобождаем память
                     lineDeallocateCall(hCall);
                     MemoLog.Lines.Add('Вызов завершен.');
                     BitBtnAnswer.Enabled := False;
                  end;
            end;
            // Приложение установило успешное соединение и готово
            // посылать данные
            LINECALLSTATE_CONNECTED:
            Begin
                  IF hcall <> 0 then
                  begin
                     Str := 'Соединение установлено...';
                     // Определяем параметры соединения
                     Callinfo.dwTotalSize := 1024;
                     // Получаем информацию
                     if lineGetCallInfo(hCall, callinfo) = 0 then
                        if callinfo.dwAppNameSize > 0 then
                            {$IFDEF WIN32}
```

```
                        Str := Str + ' ( ' + (buf +
                                callinfo.dwAppNameOffset) + ' )';
                      {$ELSE}
                        Str := Str + ' ( ' +  StrPas((buf +
                                callinfo.dwAppNameOffset)) + ' )';
                      {$ENDIF}
                        MemoLog.Lines.Add(Str);
                    end;
                end;

        LINECALLSTATE_DISCONNECTED:
        begin
                Str := 'Разъединение : ';
                // Причина ?
                if dwParam2 = LINEDISCONNECTMODE_NORMAL then
                    Str := Str + '(штатное)'
                else
                    if dwParam2 = LINEDISCONNECTMODE_BUSY then
                       Str := Str + '(занято)';
                MemoLog.Lines.Add(Str);
                BitBtnAnswer.Enabled := False;
                BitBtnDropCallClick(nil);
        end;
      end;
    end;
  end;    // with
end;
```

Функции и структуры TAPI, управляющие приемом вызовов

В данном разделе мы приводим функции и структуры TAPI, которые используются для управления приемом вызовов в TAPI.

Функция *lineAccept*

Эта функция принимает указанный вызов. Она также может послать информацию пользовательского режима к вызывающему абоненту. Данная информация обычно используется в цифровых сетях комплексного обслуживания. Также вы можете использовать ее в обычных телефонных сетях.

Когда приходит входящий вызов, он первоначально находится в состоянии "offering". В течение краткого периода времени приложение может отклонить вызов, используя функцию `lineDrop`, переадресовать, используя

lineRedirect, либо ответить на вызов, используя данную функцию. После того, как вызов был принят приложением, состояние его изменяется на "accepting" (принято). При таком состоянии приложение может также посылать информацию пользовательского режима.

Функция имеет следующий синтаксис:

```
function lineAccept(hCall: HCALL;
    lpsUserUserInfo: LPCSTR; dwSize: DWORD): Longint; stdcall;
```

Параметры функции представлены в табл. 2.21.

Таблица 2.21. *Параметры функции* `lineAccept`

Параметр	Описание
hCall	Дескриптор вызова, который должен быть принят или отклонен. Приложение должно быть владельцем вызова, и вызов должен находиться в состоянии "offering"
lpsUser UserInfo	Указатель на строку, содержащую информацию пользовательского режима, которая будет послана вызывающей стороне как часть ввода вызова. Этот указатель может быть установлен в Nil, если никакую информацию пользовательского режима передавать не нужно. Информация пользовательского режима может быть отослана только при условии, что отсылка поддерживается основной сетью *(см. структуру LINEDEVCAPS в разд. "Определение способностей телефонии")*. Поле дискриминатора протокола для информации пользовательского режима должно быть указано в первом байте строки данных. Размер данных должен быть задан в параметре dwSize
dwSize	Размер данных информации пользовательского режима

Функция возвращает положительный идентификатор вызова, если функция будет закончена асинхронно, или отрицательный код ошибки,— если произошла ошибка. Параметр dwParam2 корреспондирующего сообщения LINE_REPLY будет иметь значение ноль, если функция успешна, или отрицательное значение, — если произошла ошибка.

Функция *lineAnswer*

Эта функция отвечает на указанный вызов.

Функция имеет следующий синтаксис:

```
function lineAnswer(hCall: HCALL;
    lpsUserUserInfo: LPCSTR; dwSize: DWORD): Longint; stdcall;
```

Параметры функции те же, что и у функции `lineAccept`, за одним исключением: вызов `hCall` может находиться не только в состоянии "offering", но в состоянии "accepted".

Функция возвращает положительный идентификатор вызова, если функция будет закончена асинхронно, или отрицательный код ошибки, — если произошла ошибка. Параметр `dwParam2` корреспондирующего сообщения `LINE_REPLY` будет иметь значение ноль, если функция успешна, или отрицательное значение, — если произошла ошибка.

Функция *lineDeallocateCall*

Данная функция освобождает заданный в `hCall` дескриптор вызова. Приложения, имеющие привилегию `LINECALLPRIVILEGE_MONITOR` (мониторинг), могут безболезненно использовать данную функцию. Приложения же, имеющие привилегию `LINECALLPRIVILEGE_OWNER`, обязаны перед вызовом функции проверять, что данный вызов находится в состоянии "Idle", и приложение является единственным владельцем вызова.

Функция имеет следующий синтаксис:

```
function lineDeallocateCall(hCall: HCALL): Longint; stdcall;
```

При успешном завершении функция возвращает нуль. При ошибке — отрицательный код ошибки.

Функция *lineDrop*

Эта функция отклонить или разъединить указанный вызов. Приложение может определить информацию пользовательского режима, которая будет передана, как часть запроса "disconnect".

Функция имеет следующий синтаксис:

```
function lineDrop(hCall: HCALL;
    lpsUserUserInfo: LPCSTR; dwSize: DWORD): Longint; stdcall;
```

Параметры функции представлены в табл. 2.22.

Таблица 2.22. *Параметры функции* `lineDrop`

Параметр	Описание
`hCall`	Дескриптор вызова, который должен быть разъединен или отклонен. Приложение должно быть владельцем вызова, и вызов должен находиться в любом состоянии, кроме состояния "Idle"

Таблица 2.22 *(окончание)*

Параметр	Описание
lpsUser UserInfo	Указатель на строку, содержащую информацию пользовательского режима, которая будет послана вызывающей стороне как часть ввода запроса. Этот указатель может быть установлен в Nil, если никакую информацию пользовательского режима передавать не нужно. Информация пользовательского режима может быть отослана только при условии, что отсылка поддерживается основной сетью *(см. структуру LINEDEVCAPS в разд. "Определение способностей телефонии")*. Поле дискриминатора протокола для информации пользовательского режима должно быть указано в первом байте строки данных. Размер данных должен быть задан в параметре dwSize
dwSize	Размер данных информации пользовательского режима

Функция возвращает положительный идентификатор вызова, если функция будет закончена асинхронно, или отрицательный код ошибки, — если произошла ошибка. Параметр dwParam2 корреспондирующего сообщения LINE_REPLY будет иметь значение ноль, если функция была бы успешна, или отрицательное значение, — если произошла ошибка.

Функция *lineGetCallInfo*

Функция дает возможность приложению получить информацию о вызове.

Функция имеет следующий синтаксис:

```
function lineGetCallInfo(hCall: HCALL;
         lpCallInfo: PLineCallInfo): Longint; stdcall;
```

Параметры функции представлены в табл. 2.23.

Таблица 2.23. *Параметры функции* `lineGetCallInfo`

Параметр	Описание
hCall	Дескриптор вызова
lpCallInfo	Указатель на структуру данных LINECALLINFO. Если функция завершится удачно, то данная структура будет заполнена связанной с вызовом информацией. Прежде чем вызывать функцию lineGetCallInfo, вы должны выделить необходимый блок памяти с запасом. Приложение должно указать в поле dwTotalSize структуры LINECALLINFO размер объема памяти, доступный TAPI для возврата информации

При успешном завершении функция возвращает ноль. При ошибке — отрицательный код ошибки.

Структура *LINECALLINFO*

Огромная структура LINECALLINFO содержит информацию о вызове. Информация является фиксированной для определенного вызова. Многие функции TAPI используют LINECALLINFO в своей работе, в частности: lineGetCallInfo, TSPI_lineGetCallInfo.

Если часть структуры изменяется, приложению посылается сообщение LINE_CALLINFO. Произошедшие изменения, например изменение состояния вызова, можно определить с помощью функции lineGetCallStatus, которая возвращает структуру LINECALLSTATUS.

Если ваше приложение использует устройство-специфичные расширения (device-specific extensions), вы должны использовать поля dwDevSpecificSize и dwDevSpecificOffset, ссылающиеся на области памяти, хранящие устройство-специфичные расширения.

Поля dwCallTreatment через dwReceivingFlowspecOffset являются доступными приложениям, которые открывают устройство линии с версией TAPI 2.0 или выше.

> ***На заметку***
>
> Предпочтительный формат хранения номера для поля dwCallID и других подобных полей (dwCallerIDFlag, dwCallerIDSize, dwCallerIDOffset, dwCallerIDNameSize и dwCallerIDNameOffset) — канонический.
>
> Например, ICLID "4258828080", полученный от коммутатора должен быть преобразован в " +1 (425) 8828080", прежде чем помещен в структуру LINECALLINFO. Такое требование стандартизации облегчает работу функций повторного вызова и приложений, работающих с базами данных.

Так как структура имеет очень много параметров, приводить вид структуры мы не будем, а перейдем сразу к описанию полей. Итак, поля структуры LINECALLINFO представлены в табл. 2.24.

Таблица 2.24. *Поля структуры LINECALLINFO*

Поле	Описание
dwTotalSize	Полный размер (в байтах) структуры данных
dwNeededSize	Размер (в байтах) возвращенной информации
dwUsedSize	Размер (в байтах) той части структуры данных, которая содержит полезную информацию
hLine	Дескриптор устройства линии, с которым связан вызов

Таблица 2.24 *(продолжение)*

Поле	Описание
dwLineDeviceID	Идентификатор устройства линии, с которым связан вызов
dwAddressID	Это поле указывает адресный идентификатор, постоянно связанный с адресом вызова
dwBearerMode	Это поле указывает режим работы канала вызова. Данный параметр использует одну из констант LINEBEARERMODE_
dwRate	Скорость потока данных вызова (бит/с)
dwMediaMode	Это поле указывает тип режима носителей информационного потока. Данный параметр использует константы LINEMEDIAMODE_
dwAppSpecific	Приложение-владелец вызова, в данное поле можно установить любое значение с помощью функции lineSetAppSpecific. Поле не интерпретируется TAPI и поставщиком услуг
dwCallID	Уникальный идентификатор вызова, назначенный коммутатором или поставщиком услуг
dwRelatedCallID	Это поле указывает связанный идентификатор вызова. В средах телефонии можно связывать один вызов с другим
dwCallParamFlags	В данном поле хранится коллекция параметров исходящего вызова. Это те же самые параметры вызова, что указываются в lineMakeCall. Параметры определены константами LINECALLPARAMFLAGS_
dwCallStates	Это поле указывает состояние вызова. В данном поле может быть включено несколько битовых флагов. Данные флаги задаются константами LINECALLSTATE_. Поле dwCallStates является постоянным в LINECALLINFO и не изменяется в зависимости от состояния вызова
dwMonitor DigitModes	Это поле указывает различные цифровые режимы, заданные битовыми флагами. Флаги определены в константах LINEDIGITMODE_
dwMonitor MediaModes	Это поле указывает различные типы носителей, для которых позволен мониторинг. Типы носителей заданы битовыми флагами. Флаги определены в константах LINEMEDIAMODE_
DialParams	Это поле указывает параметры набора номера, в настоящее время действующие на вызове типа LINEDIALPARAMS. Если эти параметры не установлены ни функцией lineMakeCall, ни lineSetCallParams, то их значения такие же, как значения, по умолчанию используемые в LINEDEVCAPS-структуре
dwOrigin	Это поле указывает идентификатор, где был инициирован вызов. Поле использует одну из констант LINECALLORIGIN_

Таблица 2.24 *(продолжение)*

Поле	Описание
dwReason	Это поле указывает причину вызова. Поле использует одну из констант LINECALLREASON_
dwCompletionID	Это поле указывает идентификатор завершения для входящего вызова. Данный идентификатор значим, только если dwReason LINECALLREASON_CALLCOMPLETION
dwNumOwners	Это поле указывает количество прикладных программ, имеющих данный дескриптор вызова, открытых на линии с привилегией владельца (LINECALLPRIVILEGE_OWNER)
dwNumMonitors	Это поле указывает количество прикладных программ, имеющих данный дескриптор вызова, открытых на линии с привилегией мониторинга (LINECALLPRIVILEGE_MONITOR)
dwCountryCode	Код страны — получателя вызова. Если 0, то страна неизвестна
dwTrunk	Это поле указывает номер междугородной линии связи, по которой вызов был направлен. Этот поле используется и для входящих, и исходящих вызовов
dwCallerIDFlags	Это поле содержит флаги, подтверждающие законность (валидность) вызывающей программы или отправителя информации идентификатора вызывающей стороны. Данное поле использует одну из констант LINECALLPARTYID_
dwCallerIDSize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей идентификатор вызывающей стороны
dwCallerIDOffset	Соответствующее смещение от начала структуры LINECALLINFO
dwCallerIDName Size	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей название вызывающей стороны
dwCallerID NameOffset	Соответствующее смещение от начала структуры LINECALLINFO
dwCalledIDFlags; dwCalledIDSize; dwCalledID Offset; dwCalledIDName Size; dwCalledID NameOffset	Аналогично, только содержится информация о вызываемой стороне

Таблица 2.24 *(продолжение)*

Поле	Описание
`dwConnected IDFlags;` `dwConnected IDSize;` `dwConnected IDOffset;` `dwConnected IDNameSize;` `dwConnected IDNameOffset`	Аналогично, только содержится информация о фактически связанной стороне. Связанная сторона — это сторона, которая была фактически связана с вызовом. Информация о связанной стороне может отличаться от информации о вызываемой стороне. Такое происходит, если, например, вызов был отклонен
`dwRedirection IDFlags;` `dwRedirection IDSize;` `dwRedirection IDOffset;` `dwRedirection IDNameSize;` `dwRedirection IDNameOffset`	Аналогично, только содержится информация о стороне переназначения. Сторона переназначения — это сторона, для которой вызов был переадресован
`dwRedirecting IDFlags;` `dwRedirecting IDSize;` `dwRedirecting IDOffset;` `dwRedirecting IDNameSize;` `dwRedirecting IDNameOffset`	Аналогично, только содержится информация о стороне переадресации. Сторона переадресации — это сторона, которая переадресовала данный вызов
`dwAppNameSize`	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей название приложения, инициировавшего вызов. Имя приложения задается в функции `lineInitializeEx`. Если приложение не задает имя, то используется имя файла
`dwAppNameOffset`	Соответствующее смещение от начала структуры `LINECALLINFO`
`DwCalledParty Size`	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей описание вызываемого абонента. Данная информация может быть определена в функции `lineMakeCall` или произвольно в параметре `lpCallParams` при создании вызова

Таблица 2.24 *(продолжение)*

Поле	Описание
dwCalledParty Offset	Соответствующее смещение от начала структуры LINECALLINFO
dwCommentSize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей комментарий. Данная информация может быть определена в функции lineMakeCall или произвольно в параметре lpCallParams при создании вызова
dwComment Offset	Соответствующее смещение от начала структуры LINECALLINFO
dwDisplaySize	Это поле указывает размер (в байтах) области памяти переменного размера. Данная область содержит необработанную информацию, которая должна быть выведена на экран пользователя. В зависимости от среды телефонной связи, поставщик услуг может извлекать функциональную информацию из этого поля
dwDisplay Offset	Соответствующее смещение от начала структуры LINECALLINFO
dwUserUser InfoSize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей информацию о пользовательском режиме
dwUserUserInfo Offset	Соответствующее смещение от начала структуры LINECALLINFO
dwHighLevel CompSize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей информацию о совместимости высокого уровня. Формат этой информации определен по сопутствующим стандартам, например по стандарту ISDN Q.931
dwHighLevel CompOffset	Соответствующее смещение от начала структуры LINECALLINFO
dwLowLevel CompSize	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей информацию о совместимости нижнего уровня. Формат этой информации определен по сопутствующим стандартам, например по стандарту ISDN Q.931
dwLowLevel CompOffset	Соответствующее смещение от начала структуры LINECALLINFO
dwChargingInfo Size	Это поле указывает размер (в байтах) области памяти переменного размера. Данная область содержит так называемую управляющую информацию. Формат этой информации определен в других стандартах, например в стандарте ISDN Q.931
dwChargingInfo Offset	Соответствующее смещение от начала структуры LINECALLINFO

Таблица 2.24 *(продолжение)*

Поле	Описание
dwTerminal ModesSize	Это поле указывает размер (в байтах) области памяти переменного размера. Данная область определяет возможности оконечного устройства линии. Данные в области представленные в массиве типа DWORD. Массив имеет размерность от 0 до dwNumTerminals-1. Каждый элемент в массиве определяет текущий терминальный режим и задан соответствующими константами LINETERMMODE_
dwTerminal ModesOffset	Соответствующее смещение от начала структуры LINECALLINFO
dwDevSpecific Size	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей устройство-специфичные расширения (device-specific extensions) или, другими словами, устройство-специфичную информацию
dwDevSpecific Offset	Соответствующее смещение от начала структуры LINECALLINFO
DwCallTreatment	Это поле указывает применяемый в настоящее время обработчик вызова. Обработчик вызова применяется при изменении соответствующих состояний. Может быть нулевым, если обработчик вызова не поддержан
dwCallDataSize	Это поле указывает размер (в байтах) прикладных данных, первоначально присоединенных к вызову
dwCallData Offset	Соответствующее смещение от начала структуры LINECALLINFO
dwSending FlowspecSize	Данное поле содержит полный размер (в байтах) структуры FLOWSPEC, определенной в WinSock2 для QOS (Quality Of Service). Данная структура определяет качество обслуживания, желательное при установке исходящего соединения. Определенная поставщиком услуг часть данных после структуры FLOWSPEC не должна содержать указатели на другие блоки памяти, поскольку TAPI не сможет собрать данные, ссылки на которых заданы в указателях, и передать эти данные через взаимодействие процессов к приложению
dwSending FlowspecOffset	Соответствующее смещение от начала структуры LINECALLPARAMS
dwReceiving FlowspecSize	Данное поле содержит полный размер (в байтах) структуры FLOWSPEC, определенной в WinSock2 для QOS (Quality Of Service). Данная структура определяет качество обслуживания, желательное при установке входящего (принимающего) соединения. Определенная поставщиком услуг часть данных после структуры FLOWSPEC не должна содержать указатели на другие блоки памяти, поскольку TAPI не сможет собрать данные, ссылки на которые заданы в указателях, и передать эти данные через взаимодействие процессов к приложению

Таблица 2.24 *(окончание)*

Поле	Описание
dwReceiving FlowspecOffset	Соответствующее смещение от начала структуры LINECALLPARAMS
dwAddressType	Данное поле содержит адресный тип, используемый для вызова. Это поле доступно при использовании версии TAPI 3.0 или выше. Возможные адресные типы: LINEADDRESSTYPE_PHONENUMBER — указывает стандартный номер телефона; LINEADDRESSTYPE_SDP — указывает протокол сеанса связи; LINEADDRESSTYPE_EMAILNAME — указывает имя домена; LINEADDRESSTYPE_IPADDRESS — указывает адрес IP

Функция *lineGetCallStatus*

Эта функция возвращает текущее состояние указанного вызова.

Функция имеет следующий синтаксис:

```
function lineGetCallStatus(hCall: HCALL;
        lpCallStatus: PLineCallStatus): Longint;stdcall;
```

Параметры функции представлены в табл. 2.25.

Таблица 2.25. *Параметры функции* `lineGetCallStatus`

Параметр	Описание
hCall	Дескриптор вызова
lpCallStatus	Указатель на структуру данных LINECALLSTATUS. Если функция завершится удачно, то данная структура будет заполнена связанной с вызовом статусной информацией. Прежде чем вызывать функцию, вы должны выделить необходимый блок памяти с запасом. Приложение должно указать в поле dwTotalSize структуры LINECALLSTATUS размер объема памяти, доступный TAPI для возврата информации

При успешном завершении функция возвращает ноль. При ошибке — отрицательный код ошибки.

Структура *LINECALLSTATUS*

Структура LINECALLSTATUS описывает текущее состояние вызова. Данная информация возвращается функцией lineGetCallStatus и зависит, в свою очередь, от способностей устройства, режима вызова и текущего состояния.

Устройство-специфичные расширения, предоставленные данной структуре, используют следующие поля: dwDevSpecificSize и dwDevSpecificOffset, задающие размер и смещение выделенной для хранения области расширенных данных.

При изменении состояний вызова приложению посылается соответствующее сообщение LINE_CALLSTATE. Данное сообщение передает только основную информацию об изменившемся состоянии. Для получения дополнительной информации о состоянии в TAPI предусмотрена функция lineGetCallStatus.

Стоит также обратить внимание на то, что поля dwCallFeatures2 и tStateEntryTime доступны только приложениям, которые открывают устройство линии с версией TAPI 2.0 или выше.

Структура LINECALLSTATUS определена следующим образом:

```
PLineCallStatus = ^TLineCallStatus;
 linecallstatus_tag = packed record
 dwTotalSize,
 dwNeededSize,
 dwUsedSize,
 dwCallState,
 dwCallStateMode,
 dwCallPrivilege,
 dwCallFeatures,
 dwDevSpecificSize,
 dwDevSpecificOffset: DWORD;
{$IFDEF TAPI20}
 dwCallFeatures2: DWORD; // TAPI v2.0
{$IFDEF WIN32}
 tStateEntryTime: TSystemTime; // TAPI v2.0
{$ELSE}
 tStateEntryTime: array[0..7] of WORD; // TAPI v2.0
{$ENDIF}
{$ENDIF}
end;
TLineCallStatus = linecallstatus_tag;
LINECALLSTATUS = linecallstatus_tag;
```

Поля структуры LINECALLINFO представлены в табл. 2.26.

Таблица 2.26. *Поля структуры* `LINECALLINFO`

Поле	Описание
`dwTotalSize`	Полный размер (в байтах) структуры данных
`dwNeededSize`	Размер (в байтах) возвращенной информации
`dwUsedSize`	Размер (в байтах) той части структуры данных, которая содержит полезную информацию
`dwCallState`	Определяет текущее состояние вызова. Поле использует константы `LINECALLSTATE_`
`dwCallStateMode`	Определяет дополнительный режим состояния вызова. Поле использует константы `LINECALLSTATE_`
`dwCallPrivilege`	Поле определяет привилегию вызова, определенную в константах `LINECALLPRIVILEGE_`
`dwCallFeatures`	Данное поле содержит битовые флаги, указывающие, какие функции TAPI сможет использовать приложение на данном вызове в текущем состоянии. Если в определенном флаге стоит 0, то соответствующая функция не может быть использована приложением в данном вызове в текущем состоянии. Поле использует константы `LINECALLFEATURE_`, описание которых вы можете найти в файле помощи по TAPI
`dwDevSpecificSize`	Это поле указывает размер (в байтах) области памяти переменного размера, содержащей устройство-специфичные расширения
`dwDevSpecificOffset`	Это поле указывает смещение, заданное от начала структуры области памяти переменного размера, содержащей устройство-специфичные расширения
`dwCallFeatures2`	Аналогично `dwCallFeatures`. Поле содержит дополнительный набор битовых флагов, заданных в константах `LINECALLFEATURE2_`
`tStateEntryTime`	Время изменения состояния

Заключение

В данной главе мы представили основные навыки работы с TAPI. Конечно, приведенная здесь информация явно недостаточна для углубленного исследования данного программного интерфейса. Однако приведенный материал позволит понять основные концепции и механизмы TAPI и послужит толчком к более подробному изучению.

ГЛАВА 3

MCI-интерфейс для работы с мультимедиа

Введение

Media Control Interface (MCI) — это интерфейс прикладного программирования, предназначенный для управления различными мультимедийными устройствами в Windows. В частности, с помощью данного интерфейса прикладной программист получает в свои руки возможность управления такими устройствами, как Wave-звуковые, MIDI-синтезаторы, CD- и DVD-ROM, а также устройства воспроизведения и записи цифрового видео. Прелесть данного интерфейса в том, что он является единым для всех мультимедийных устройств.

Абстрагируясь от низкоуровневых команд управления, MCI предоставляет высокоуровневый интерфейс с единой системой команд, независимых от различных мультимедийных устройств. MCI — довольно известный интерфейс, поддерживающийся почти всеми существующими звуковыми и видеоаппаратными средствами ЭВМ. Стоит также заметить, что MCI является частью Win32 Multimedia API, который, в свою очередь, предоставляет более широкие возможности по работе с мультимедиа. Для работы с данным интерфейсом вам нужно подключить модуль `MMSystem` (входящий в стандартную поставку Delphi), в котором описывается большинство MCI-функций.

К сожалению, объем данной главы не представляет возможности подробно остановиться на всех тонкостях работы с MCI, поэтому здесь мы приведем наиболее интересные моменты использования данного API с точки зрения прикладного программиста.

Интерфейс командных строк и команд-сообщений MCI

Идеология работы MCI в большей своей части основана на использовании так называемых интерфейсов командных строк (command strings) и команд-

сообщений (command messages). Интерфейс командных строк удобен для быстрого получения необходимых результатов, а интерфейс команд-сообщений — для создания сложных мультимедийных приложений. Данные интерфейсы выполняют примерно одни и те же функции. Отличие между этими двумя интерфейсами заключается только в способе подачи команд. Прежде чем приступить к разбору MCI, более подробно остановимся на этом вопросе.

Командные строки

Каждое мультимедийное устройство имеет свой внутренний набор команд. Посылка командной строки MCI с помощью функции `mciSendString` приведет к выполнению тех или иных действий. Например, командная строка '`Set cdaudio door open wait`' указывает устройству CD-ROM (в MCI: cdaudio) открыть лоток, а строка '`Set cdaudio door closed wait`', соответственно, закрыть его. Для посылки командных строк в MCI определена функция `mciSendString`, имеющая следующий вид:

```
function mciSendString(lpstrCommand,
            lpstrReturnString: PChar;
            uReturnLength: UINT;
            hWndCallback: HWND): MCIERROR;
```

В качестве параметров функция принимает следующие аргументы, представленные в табл. 3.1.

Таблица 3.1. *Параметры* `mciSendString`

Параметр	Описание
`lpstrCommand`	Командная строка для устройств MCI
`lpstrReturnString`	Строка возврата информации об исполнении команды
`uReturnLength`	Размер буфера исполняемой строки
`hWndCallback`	Хэндл (Handle) окна для приема MCI-сообщений. Данный параметр используется, если в вашей программе присутствуют процедуры обработки MCI-приложений

После завершения функция возвращает код ошибки `MCIERROR`. Если функция была выполнена успешно, то возвращаемое значение равно 0; если при выполнении команды произошла ошибка, то `MCIERROR` вернет код ошибки, причем в младшем слове `MCIERROR` будет непосредственно код ошибки, а в старшем слове возвращаемой величины — идентификатор драйвера, если ошибка

вызвана устройством. Для разбора ошибки и получения текстового сообщения об ошибке можно использовать функцию MCI `mciGetErrorString`.

Командные строки MCI состоят из последовательного описания идентификатора устройства и команд управления. В некоторых случаях команды управления содержат дополнительные аргументы, предназначенные для конкретной команды.

Командная строка имеет следующий вид: `"command device_id arguments"`, где `command` — команда, `device_id` — идентификатор MCI-устройства, `arguments` — дополнительные аргументы команды. Идентификаторы MCI-устройств представлены в табл. 3.3.

Некоторые команды MCI, такие как `open`, `close` или `play`, имеют дополнительные аргументы. Команда `play` имеет аргументы `"from position"` и `"to position"`, указывающие на то, с какой и по какую позицию нужно проиграть мультимедийный файл. Стоит также отметить, что если вы не указываете аргументы в строке команды, то команда примет аргументы по умолчанию. Также в командах в качестве переменных можно использовать следующие типы данных: `String`, `Integer` и `TRect`.

Пример использования командной строки, открывающей лоток CD-ROM:

```
mciSendString('Set cdaudio door open wait', nil, 0,
        application.handle);
```

Команды сообщений

MCI помимо обычных командных строк поддерживает так называемые команды сообщений (command messages). Команды сообщений MCI состоят из трех элементов:

- код сообщения;
- структура, содержащая параметры для команды;
- набор флагов, определяющих варианты для команды и утверждающий поля в блоке параметров.

Для посылки команд сообщений в MCI определена функция `mciSendCommand`, имеющая следующий вид:

```
function mciSendCommand(mciId: MCIDEVICEID;
            uMessage: UINT;
            fdwCommand, dwParam: DWORD): MCIERROR;
```

В качестве параметров функция принимает следующие аргументы, представленные в табл. 3.2.

Таблица 3.2. Параметры `mciSendCommand`

Параметр	Описание
`mciId`	Идентификатор устройства MCI. Этот параметр не используется с командой сообщения `MCI_OPEN`. Константы идентификаторов MCI устройств вы можете просмотреть в табл. 3.3
`uMessage`	Команда сообщения
`fdwCommand`	Флаги для команды сообщения. Флаги — это битовые матрицы, складываемые оператором `or`
`dwParam`	Указатель на адрес структуры, которая содержит параметры для команды сообщения. Поле `dwCallback` данной структуры идентифицирует окно, которое получает `MM_MCINOTIFY`-сообщение, когда действие команды заканчивается

После завершения функция возвращает код ошибки `MCIERROR`. Если функция выполнена успешно, то возвращаемое значение равно 0; если при выполнении команды произошла ошибка, то `MCIERROR` вернет код ошибки. Для разбора ошибки и получения текстового сообщения об ошибке можно использовать функцию MCI `mciGetErrorString`.

Пример использования нескольких команд-сообщений, работающих с CD-ROM, представлен в листинге 3.1. В данном примере программа произведет открытие лотка CD-ROM.

Листинг 3.1. Управление лотком CD-ROM

```
var
  mciOpenParms : MCI_OPEN_PARMS;
  dwDeviceID   : DWORD;
begin
  mciOpenParms.lpstrDeviceType := PChar(MCI_DEVTYPE_CD_AUDIO);
  if mciSendCommand(0, MCI_OPEN, MCI_OPEN_TYPE or
        MCI_OPEN_TYPE_ID, DWORD(@mciOpenParms)) <> 0 then Exit;
  dwDeviceID := mciOpenParms.wDeviceID;
  mciSendCommand(dwDeviceID, MCI_SET, MCI_SET_DOOR_OPEN, 0);
  mciSendCommand(dwDeviceID, MCI_CLOSE, 0, 0);
  Halt(0);
end.
```

Типы и драйверы MCI-устройств

В предыдущих разделах мы разобрали два интерфейса, предназначенных для посылки управляющих команд MCI, однако вопрос взаимодействия операционной системы через MCI с конкретными драйверами мультимедийных уст-

ройств остался открытым. В данном разделе мы попробуем прояснить данный вопрос. При установке драйверов, поддерживающих стандарт MCI, делается соответствующая запись в файле SYSTEM.INI и в соответствующем разделе реестра Windows, которая связывает идентификатор MCI-устройства с конкретным драйвером. Часть драйверов, поддерживающих MCI, инсталлируется при установке Windows, часть — при последующей установке различных устройств и программ.

На рис. 3.1 вы можете просмотреть, как выглядит связка идентификаторов и драйверов устройств в реестре Windows.

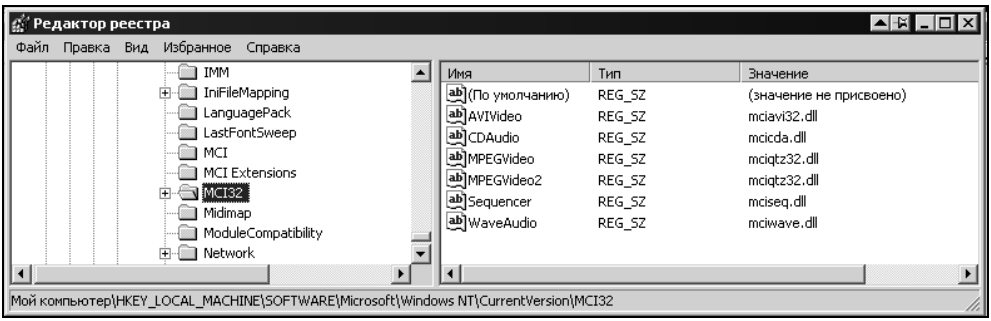


Рис. 3.1. Связка идентификаторов MCI с драйверами устройств

Как вы заметили, с одним и тем же идентификатором MCI-устройства может быть связано несколько разных драйверов. В этом случае для идентификации MCI-устройства с конкретным драйвером к идентификатору MCI-устройства добавляется номер устройства.

В табл. 3.3 представлены основные задокументированные идентификаторы MCI-устройств. Следует также заметить, что идентификатор устройства используется в командных строках, а константа устройства — в интерфейсах команд сообщений.

Таблица 3.3. *Константы и идентификаторы устройств MCI*

Идентификатор устройства для командных строк	**Константа для команд-сообщений**	**Описание MCI-устройства**
`Нет`	`MCI_ALL_DEVICE_ID`	Все устройства
`Нет`	`MCI_DEVTYPE_ANIMATION`	Устройство проигрывания анимационных файлов (Animation-playback device)
`cdaudio`	`MCI_DEVTYPE_CD_AUDIO`	Проигрыватель аудиокомпакт-дисков (CD audio device)

Таблица 3.3 *(окончание)*

Идентификатор устройства для командных строк	Константа для команд-сообщений	Описание MCI-устройства
dat	MCI_DEVTYPE_DAT	Цифровой аудиопроигрыватель (Digital-audio tape device)
digitalvideo	MCI_DEVTYPE_DIGITAL_VIDEO	Цифровой видеопроигрыватель (Digital-video playback device)
other	MCI_DEVTYPE_OTHER	Неопределенное устройство
overlay	MCI_DEVTYPE_OVERLAY	Проигрыватель аналогового видео в окне (оверлейное видео) (Video-overlay device)
scanner	MCI_DEVTYPE_SCANNER	Управление сканером (Scanner device)
sequencer	MCI_DEVTYPE_SEQUENCER	MIDI-синтезатор (MIDI sequencer device)
vcr	MCI_DEVTYPE_VCR	Управление видеомагнитофоном (Video-cassette recorder)
videodisc	MCI_DEVTYPE_VIDEODISC	VCD-проигрыватель (Videodisc player)
waveaudio	MCI_DEVTYPE_WAVEFORM_AUDIO	Проигрывание/запись WAVE-файлов (Waveform-audio device)

Помимо основных идентификаторов могут появляться также и дополнительные идентификаторы, такие как `avivideo` — проигрыватель AVI-файлов, `qtwvideo` — проигрыватель Quick Time for Windows для видеофайлов в формате Quick Time (*.mov).

Классификация MCI-команд

Команды MCI классифицируются по четырем категориям.

- *Системные команды* (System commands) — обрабатываются непосредственно MCI без участия драйверов мультимедийных устройств.
- *Обязательные команды* (Required commands) — непосредственно обрабатываются конкретным драйвером мультимедийного устройства. По спецификации любые MCI-совместимые драйверы и устройства должны поддерживать обязательные команды и флаги.

- *Основные или опциональные команды* (Basic commands or optional commands) — служат для управления MCI-устройствами. Предназначены также для установки дополнительных опций основных команд путем поддержки определенных наборов флагов для той или иной команды.
- *Расширенные команды* (Extended commands) — являются действенными только к определенному типу устройства или драйверу.

В то время как системные и обязательные команды являются минимальным набором для любого MCI-устройства, основные и расширенные команды не поддерживаются всеми MCI-устройствами. Ваше приложение может всегда использовать системные или обязательные команды, однако перед использованием основных или расширенных команд вам нужно для начала определить, поддерживает ли MCI-устройство данную команду. Для определения следует воспользоваться требуемой командой, сообщением `MCI_GETDEVCAPS`.

В табл. 3.4 представлена классификация системных, требуемых и основных команд MCI, разбитых по категориям.

Таблица 3.4. *Классификация команд MCI*

Командная строка	Команда сообщения	Описание команды
СИСТЕМНЫЕ КОМАНДЫ (System commands)		
`break`	`MCI_BREAK`	Назначение виртуального кода клавиши, с помощью которой можно прервать работу MCI-устройства
`sysinfo`	`MCI_SYSINFO`	Возвращает системную информацию о MCI- устройстве в виде текстовой строки
ОБЯЗАТЕЛЬНЫЕ КОМАНДЫ (Required commands)		
`capability`	`MCI_GETDEVCAPS`	Получает информацию о возможностях MCI-устройства
`close`	`MCI_CLOSE`	Закрывает MCI-устройство
`info`	`MCI_INFO`	Получает текстовую информацию от MCI-устройства
`open`	`MCI_OPEN`	Первичная инициализация и открытие устройства
`status`	`MCI_STATUS`	Получает статусную информацию от MCI-устройства
ОСНОВНЫЕ КОМАНДЫ (Basic commands)		
`Load`	`MCI_LOAD`	Загрузка данных из файла
`Pause`	`MCI_PAUSE`	Пауза. Воспроизведение и запись могут быть возобновлены из текущей позиции мультимедийных данных

Таблица 3.4 *(окончание)*

Командная строка	Команда сообщения	Описание команды
Play	MCI_PLAY	Запуск воспроизведения
Record	MCI_RECORD	Старт записи входящих данных
Resume	MCI_RESUME	Продолжение воспроизведения или записи
Save	MCI_SAVE	Запись мультимедийных данных на диск
Seek	MCI_SEEK	Позиционирование (перемотка) в мультимедийных данных
Set	MCI_SET	Устанавливает операционное (режимное) состояние устройства
Status	MCI_STATUS	Получает статусную информацию от MCI-устройства. Это обязательная команда, но т. к. в данной команде могут быть дополнительные флаги, она представлена и в основных командах

Как было сказано ранее, некоторые MCI-устройства предоставляют разработчику дополнительные команды или модифицируют существующие основные команды, дополняя их расширенными флагами. Все такие команды входят также в категорию расширенных команд. В то время как некоторые расширенные команды применяются только для конкретного MCI-устройства, большинство расширенных команд является модифицированными основными командами. Например, команда MCI_SET для MIDI-устройств (SEQUENCER) имеет расширенный набор флагов и попадает в расширенный набор команд для устройства SEQUENCER. Поэтому, прежде чем использовать команду из расширенного набора, следует определить, поддерживает ли данное устройство эту команду с таким набором флагов с помощью команды сообщения MCI_GETDEVCAPS.

В табл. 3.5 представлена совместимость расширенного набора команд MCI с устройствами MCI. Следует иметь в виду, что таблица расширенных команд является неполной, т. к. для специфичных MCI-устройств могут быть дополнительные (используемые только для него) команды. Например, в данной таблице нет большинства расширенных команд для цифрового видеопроигрывателя и проигрывателя AVI-файлов. Будем считать, что в данной таблице представлен основной (часто используемый) набор расширенных команд.

Таблица 3.5. *Основной набор расширенных команд MCI*

Командная строка	Команда сообщения	MCI-устройства, поддерживающие данную команду	Описание команды
configure	MCI_CONFIGURE	digitalvideo	Показ конфигурационного диалогового окна
cue	MCI_CUE	digitalvideo, waveaudio	Подготовка MCI-устройства для проигрывания или записи
delete	MCI_DELETE	waveaudio	Удаление сегмента аудиоданных из звукового файла
escape	MCI_ESCAPE	videodisc	Посылка информации устройству
freeze	MCI_FREEZE	overlay	Освобождает видеобуфер с выводом изображения Настройка дисплея
put	MCI_PUT	digitalvideo, overlay	Устанавливает источник, получатель и размеры окна для вывода видео
realize	MCI_REALIZE	digitalvideo	Устанавливает и настраивает цветовую палитру в окне вывода цифрового видеопроигрывателя
setaudio	MCI_SETAUDIO	digitalvideo	Устанавливает аудиопараметры для цифрового видеопроигрывателя
setvideo	MCI_SETVIDEO	digitalvideo	Устанавливает видеопараметры для цифрового видеопроигрывателя
signal	MCI_SIGNAL	digitalvideo	Маркирует указанную позицию в медиаданных
spin	MCI_SPIN	videodisc	Начинает вращение VCD-диска или останавливает диск
step	MCI_STEP	digitalvideo, videodisc	Пошаговое перемещение вперед или назад по медиа-контенту
unfreeze	MCI_UNFREEZE	overlay	Дает возможность буферу изображения получить видеоданные
update	MCI_UPDATE	digitalvideo	Обновляет фрейм в контексте устройства, фактически обновляет (перерисовывает) окно проигрывателя содержимым текущего кадра

Таблица 3.5 *(окончание)*

Командная строка	Команда сообщения	MCI-устройства, поддерживающие данную команду	Описание команды
where	MCI_WHERE	digitalvideo, overlay	Получает данные об источнике, получателе и размере окна для вывода видео
window	MCI_WINDOW	digitalvideo, overlay	Определяет окно, в которое необходимо вывести цифровое видеоизображение. Без этой команды изображение выводится в окно, созданное драйвером MCI и используемое по умолчанию

Далее мы немного остановимся на параметрах команд и рассмотрим их в контексте интерфейса командных строк, как более простом для изучения. Сразу оговоримся, что перевод данных параметров в интерфейс команд сообщений — не особо сложная задача и решается она обычно выставлением соответствующих флагов в структурах MCI. Так что, разбирать команды в интерфейсе команд-сообщений мы не станем, а отошлем читателя к примерам, находящимся на прилагаемом компакт-диске в каталоге Source\Ch03\.

В табл. 3.6 представлено краткое описание параметров MCI-команд. Более подробное описание параметров вы можете найти в SDK.

Синтаксис описания параметров следующий:

- `{}` — обязательные параметры;
- `|` — возможные варианты параметров конкретной команды;
- *индекс* — аргумент параметра;
- `[параметр]` — необязательный (дополнительный) параметр команды.

Таблица 3.6. *Параметры команд MCI в интерфейсе командных строк*

Команда	Параметры
break	{off | on *код клавиши*}
sysinfo	{installname | name *индекс* | name *индекс* open | quantity | quantity open}
capability	{can eject | can play | can record | can save | compound device | device type | has audio | has video | uses files}

Таблица 3.6 *(окончание)*

Команда	Параметры
info	[product]
open	[alias *псевдоним*] [shareable] [type *тип устройства*]
status	[mode | ready] В случае параметра mode обязательным набором ответов команды являются следующие значения: *not ready* — устройство не готово; *paused* — пауза; *playing* — проигрывание; *stopped* — устройство остановлено
load	{*имя файла*}
play	[from *позиция*] [to *позиция*]
record	[from *позиция*] [to *позиция*] [insert | overwrite]
save	[*имя файла*]
seek	{to *позиция*| to start | to end}
set	[audio all off | audio all on | audio left off | audio left on | audio right off | audio right on | video off | video on] [door closed | door open] [time format milliseconds | time format ms]
status	{current track | length | length track номер | mode | number of tracks | position | position track *номер* | ready | start position | time format}

Как видно из таблицы, например, для MCI-команды Seek возможны три варианта задаваемых параметров, а именно:

- to *позиция* — позиционирование в заданную позицию;
- to start — позиционирование на начало;
- to end — позиционирование на конец.

Функции и макросы MCI

Хотя большинство MCI-приложений используют только функции mciSendString и mciSendCommand, в самой MCI существуют также некоторые другие полезные функции, которыми вы будете пользоваться не менее часто. Наиболее полезные и часто используемые из них:

- mciGetDeviceID — возвращает идентификатор устройства без оного. Обычно получение идентификатора связано с использованием команды MCI_OPEN, которая, в свою очередь, открывает устройство MCI для работы.

Если же вы по каким-либо причинам не хотите открывать устройство, данная функция для вас;

- `mciGetCreatorTask` — позволяет вашему приложению использовать идентификатор устройства для получения хэндла задачи, создавшего данный идентификатор;
- `mciGetYieldProc` и `mciSetYieldProc` — используются для установки и получения адреса `callback`-функции, служащей для получения сообщений от MCI при выполнении команд с флагом `MCI_WAIT`. То есть функция `mciSetYieldProc` позволяет MCI продолжать диалог с вашим приложением после того, как вы дадите команду MCI с флагом `WAIT`. Этот флаг сообщает MCI о том, что управление должно вернуться к приложению только тогда, когда посланная команда будет полностью выполнена. Например, вы можете дать MCI команду воспроизведения MIDI-файла продолжительностью в несколько минут. Без флага `WAIT` MCI начнет воспроизводить файл и тут же вернет управление вашей программе, а сам MIDI-файл будет воспроизведен в фоне. Если же вы используете флаг `WAIT` с командой воспроизведения, то MCI вернет управление вашей программе только после доигрывания файла до конца. Однако до проигрывания вы можете определить `Callback`-функцию для поддержки диалога вашего приложения с MCI;
- `mciGetErrorString` — функция переводит код ошибки в строковое значение описания данной ошибки.

Помимо функций в MCI существуют макросы. Макросы, представленные в табл. 3.7, создают или получают формат позиционирования внутри медиа. Данные макросы очень важны при создании MCI-приложений. В настоящее время MCI поддерживает следующие форматы позиционирования:

- HMS — часы/минуты/секунды (hours/minutes/seconds);
- MSF — минуты/секунды/фреймы (minutes/seconds/frames);
- TMSF — трек/минуты/секунды/фреймы (tracks/minutes/seconds/frames).

Сразу оговоримся, не все форматы позиционирования поддерживаются различными MCI-устройствами. При попытке установить неподдерживаемый устройством формат, он будет отвергнут.

Таблица 3.7. *Макросы форматов позиционирования MCI*

Макрос	Описание
`MCI_HMS_HOUR`	Возвращает часы из формата HMS
`MCI_HMS_MINUTE`	Возвращает минуты из формата HMS
`MCI_HMS_SECOND`	Возвращает секунды из формата HMS

Таблица 3.7 *(окончание)*

Макрос	Описание
MCI_MAKE_HMS	Создает формат позиционирования HMS. Данный формат имеет тип — целое число длиной 4 байта. Значения от младшего к старшему байту: • часы (младший байт); • минуты; • секунды; • неиспользуемый (старший байт)
MCI_MAKE_MSF	Создает формат позиционирования MSF. Данный формат имеет тип — целое число длиной 4 байта. Значения от младшего к старшему байту: • минуты (младший байт); • секунды; • фреймы; • неиспользуемый (старший байт)
MCI_MAKE_TMSF	Создает формат позиционирования TMSF. Данный формат имеет тип — целое число длиной 4 байта. Значения от младшего к старшему байту: • дорожки (младший байт); • минуты; • секунды; • фреймы (старший байт)
MCI_MSF_FRAME	Возвращает фреймы из формата MSF
MCI_MSF_MINUTE	Возвращает минуты из формата MSF
MCI_MSF_SECOND	Возвращает секунды из формата MSF
MCI_TMSF_FRAME	Возвращает фреймы из формата TMSF
MCI_TMSF_MINUTE	Возвращает минуты из формата TMSF
MCI_TMSF_SECOND	Возвращает секунды из формата TMSF
MCI_TMSF_TRACK	Возвращает трек из формата TMSF

Сообщения MCI

В данном разделе мы остановимся на основных сообщениях MCI. Как вы знаете, обработка сообщений лежит в основе работы различных приложений

Windows. Сообщение в Windows — это фактически структура данных, которая передается приложению при возникновении того или иного события. Структура сообщения содержит в себе: `Hwdn` — дескриптор окна, `Message` — тип сообщения, `WParam` — значение `Wparam`, `LParam` — значение `Lparam`, а также отметку времени отправки `Time` (в миллисекундах после запуска Windows) и структуру `Point`, содержащую координаты X, Y курсора мыши. Сообщения MCI ничем не отличаются от сообщений Windows.

Следующие сообщения являются MCI-сообщениями:

- `MM_MCINOTIFY` — возникает в тот момент, когда завершена посланная ранее MCI-команда с установленным флагом `MCI_NOTIFY`. Данное сообщение передается в `Callback`-функцию, адрес которой был передан через поле `dwCallback` команды-сообщения.

  Сообщение передает следующие значения в полях:

  - `WParam` — флаг выполнения, может принимать следующие значения:
    - `MCI_NOTIFY_ABORTED` — MCI-устройство получило команду, которая предотвратила или прервала действие поданной ранее команды. Например, при воспроизведении с помощью ранее поданной команды `MC_Play` была получена команда `MC_Stop`;
    - `MCI_NOTIFY_FAILURE` — ошибка выполнения команды MCI-устройством;
    - `MCI_NOTIFY_SUCCESSFUL` — штатное (успешное) выполнение команды;
    - `MCI_NOTIFY_SUPERSEDED` — команда заменена другой.
  - `LParam` — идентификатор MCI-устройства, посылающего данное сообщение в `Callback`-функцию.

- `MM_MCISIGNAL` — определено только для цифровых видеоустройств. Это сообщение возникает в тот момент времени, когда позиция в медиаданных достигает позиции, определенной командой-сообщением `MCI_SIGNAL`.

  Сообщение передает следующие значения в полях:

  - `WParam` — идентификатор MCI-устройства, посылающего данное сообщение;
  - `LParam` — значение, переданное в поле `dwUserParm` структуры `MCI_DGV_SIGNAL_PARAMS`, т. е. позиция в медиаданных.

Пример определения `Callback`-функции для получения сообщения `MM_MCINOTIFY` приведен в листинге 3.2.

Листинг 3.2. Работа с `Callback`-функцией

```
Type
// ------------------------------------------------------------------
// Определение Callback-функции для получения сообщения
// MM_MCINOTIFY
// ------------------------------------------------------------------
procedure Message_MM_MCINOTIFY(var Message: TMessage);
                      message MM_MCINOTIFY;
// ------------------------------------------------------------------
// Пример использования MCI-команды с флагом mci_Notify
// ------------------------------------------------------------------
var
PlayParm: TMCI_Play_Parms;
FFlags: Longint;
begin
PlayParm.dwCallback := Handle;// Окно для получения сообщений
FFlags := 0;
// Посылка сообщения MM_MCINOTIFY после выполнения MCI-команды
FFlags := FFlags := FFlags or mci_Notify;
mciSendCommand(MCI_DEVTYPE_WAVEFORM_AUDIO,
        mci_Play, FFlags, Longint(@PlayParm));
end;
// ------------------------------------------------------------------
// Функция обработки сообщения MM_MCINOTIFY
// ------------------------------------------------------------------
procedure Message_MM_MCINOTIFY(var Message: TMessage);
begin
case Message.WParam of
 mci_Notify_Successful: begin
             // что-то делаем
             end;
 mci_Notify_Superseded: begin
             end;
 mci_Notify_Aborted  : begin
             end;
 mci_Notify_Failure  : begin
             end;
end;
end;
```

Общие флаги для MCI-команд

Как вы уже знаете, при посылке команд с помощью функций `mciSendCommand` и `mciSendString` вы можете задавать также различные флаги, влияющие на поведение отправляемой команды. Флаги — это битовые матрицы, складываемые оператором `or`. Для каждой команды флаги могут быть свои, однако существуют флаги, являющиеся общими для любой MCI-команды. Таких флагов два — `wait` и `notify`. Рассмотрим их подробнее, т. к. использование данных флагов пригодится нам при создании приложений.

- Флаг `wait` применяется для установки ожидания окончания выполняемого процесса приложением. Если в MCI-команде не используется данный флаг, то при выполнении данной команды MCI запускает фоновый процесс и сразу же возвращает управление вызвавшему данный процесс приложению. Что, согласитесь, не совсем удобно, например, очень сложно прервать выполнение MCI-процесса без использования клавиши прерывания <Ctrl>+<Break>. Для такой ситуации в программе обычно предусматривают средства досрочного прекращения фонового MCI-процесса с помощью других MCI-команд. Если же в команде используется флаг `wait`, то ваше приложение ждет окончания этого MCI-процесса, а после его окончания получает дальнейшее управление.
- Флаг `notify` служит для уведомления приложения о завершении MCI-процесса. Как было сказано, при использовании MCI-команд без флага `wait` запустится фоновый MCI-процесс, после чего выполнение программы и процесса будет идти параллельно. Для того чтобы программа, запустившая процесс, узнала о завершении параллельного MCI-процесса, нужно использовать флаг `notify`. Действительно, не существует более простого способа уведомить приложение о завершении MCI-процесса. Теоретически можно проверять данный процесс с помощью низкоуровневых API-функций, но такой подход является трудоемким и малоэффективным. После завершения MCI-команды с флагом `notify` приложению будет послано сообщение `MM_MCINOTIFY`, информирующее о завершении команды и возвращающее результат выполнения команды в MCI-устройстве.

Структуры данных MCI

Большинство команд-сообщений используют различные структуры данных, передаваемые в виде указателя на адрес структуры в поле `dwParam` функции `mciSendCommand`. Для различных команд-сообщений предусмотрены различные структуры. Вызывая `mciSendCommand`, мы создаем структуру с данными, необходимыми конкретной команде, и передаем функции указатель на нее. На-

пример, при отправке MCI-команды MCI_PLAY с помощью mciSendCommand в поле dwParam передается указатель на структуру MCI_PLAY_PARMS, а при посылке команды MCI_SET указатель на структуру MCI_SET_PARMS.

Стоит отметить, что некоторые MCI-команды могут принимать различные структуры в параметре dwParam. Данные структуры являются специфичными для конкретных MCI-устройств. Например, команда-сообщение MCI_OPEN может принимать структуру MCI_ANIM_OPEN_PARMS при открытии устройства проигрывания анимации или общую структуру MCI_Open_Parms.

Прежде чем использовать различные MCI-команды, следует проверить, используют ли они структуры данных в качестве параметра dwParam, а в случае использования правильно заполнить поля, представленные в данных структурах. Также следует обратить внимание на MCI-устройство, которому будет послана команда, т. к. возможно для конкретного устройства и конкретной команды существует своя собственная расширенная (по сравнению со стандартной) структура.

Следует также отметить, что все MCI-структуры имеют общее поле dwCallback, в котором указывается хэндл (Handle) окна для приема MCI-сообщений.

Также стоит иметь в виду следующее: для того чтобы MCI-команда приняла к сведению параметры, заключенные в полях данных функций, нужно установить соответствующие флаги в параметре fdwCommand функции mciSendCommand. Для каждой MCI-команды данные флаги свои. Например, для того чтобы команда MCI_STEP приняла к сведению поле dwFrames структуры MCI_ANIM_STEP_PARMS, нужно указать следующие флаги в функции mciSendCommand для следующих MCI-устройств:

- MCI_ANIM_STEP_FRAMES — для устройства проигрывания анимационных файлов (Animation);
- MCI_DGV_STEP_FRAMES — для цифровых видеоустройств (digitalvideo);
- MCI_VCR_STEP_FRAMES — для видеомагнитофона (VCR)

  и т. д.

В табл. 3.8 приведены краткие описания структур MCI, разбитых по категориям устройств. Общие структуры могут быть использованы для всех категорий устройств. Более подробную информацию вы всегда сможете найти в Multimedia Programmer's Reference.

Несмотря на большое количество представленных структур, многие из них, привязанные к MCI-устройству, являются взаимозаменяемыми с общими структурами.

Таблица 3.8. *Краткое описание структур MCI*

Структура	Описание
ОБЩИЕ СТРУКТУРЫ ДЛЯ ВСЕХ УСТРОЙСТВ	
MCI_GENERIC_PARMS	Структура, не содержащая в себе никаких параметров, кроме поля dwCallback. Служит в своем роде заглушкой для MCI-команд, которые не работают с различными дополнительными данными, представленными в других структурах
MCI_BREAK_PARMS	Структура содержит информацию о коде клавиши прерывания MCI-процесса. Данная структура предназначена для команды-сообщения MCI_BREAK. Данный код клавиши используется, если MCI-команда стартовала на выполнение с флагом MCI_BREAK_KEY. По умолчанию устанавливается комбинация клавиш <Ctrl>+<Break>
MCI_GETDEVCAPS_PARMS	Данная структура предназначена для команды-сообщения MCI_GETDEVCAPS, считывающей с MCI-устройства характерную для устройства информацию
MCI_INFO_PARMS	Данная структура предназначена для получения информации о MCI-устройстве с помощью команды-сообщения MCI_INFO
MCI_LOAD_PARMS	Данная структура содержит информацию о загружаемом в MCI-устройство файле и используется в команде-сообщении MCI_LOAD
MCI_OPEN_PARMS	Данная структура используется для команды-сообщения MCI_OPEN и содержит в себе идентификатор wDeviceID данного устройства, который, в свою очередь, будет использован для команд-сообщений
MCI_PLAY_PARMS	Данная структура содержит информацию о начальном и конечном сегменте воспроизводимых медиаданных и предназначена для команды-сообщения MCI_PLAY
MCI_RECORD_PARMS	Данная структура содержит информацию о начальной позиции записываемых медиаданных и предназначена для команды-сообщения MCI_ RECORD
MCI_SAVE_PARMS	Данная структура содержит информацию о сохраняемом с помощью команды-сообщения MCI_Save файле
MCI_SEEK_PARMS	Данная структура содержит информацию для позиционирования в мультимедийных данных с помощью команды-сообщения MCI_SEEK

Таблица 3.8 *(продолжение)*

Структура	Описание
MCI_SET_PARMS	Структура содержит данные, настраивающие опциональное состояние MCI-устройства. Используется в команде-сообщении MCI_SET
MCI_STATUS_PARMS	Данная структура предназначена для получения статусной информации об MCI-устройстве. Используется в команде-сообщении MCI_STATUS
MCI_SYSINFO_PARMS	Данная структура предназначена для получения различной технической информации об MCI-устройстве. Используется в команде-сообщении MCI_SYSINFO
СТРУКТУРЫ ДЛЯ MIDI-СИНТЕЗАТОРА	
MCI_SEQ_SET_PARMS	Структура содержит данные об устанавливаемых настройках MIDI-синтезатора, таких как темп проигрывания, выходной порт и т. д. Используется для установки опций устройства в команде-сообщении MCI_SET
СТРУКТУРЫ ДЛЯ ПРОИГРЫВАТЕЛЯ АНИМАЦИИ	
MCI_ANIM_OPEN_PARMS	Данная структура предназначена для команды-сообщения MCI_OPEN. Если вы не хотите использовать расширенные элементы данной структуры, при работе с MCI_OPEN вы можете воспользоваться структурой MCI_Open_Parms вместо MCI_Anim_Open_Parms
MCI_ANIM_PLAY_PARMS	Данная структура содержит информацию о начальном и конечном сегменте воспроизводимой анимации Предназначена для команды-сообщения MCI_PLAY. Если вы не хотите использовать расширенные элементы, такие как начало и конец воспроизведения, вы можете воспользоваться структурой MCI_Play_Parms
MCI_ANIM_RECT_PARMS	Данная структура содержит информацию о размерах окна для команд-сообщений MCI_PUT и MCI_WHERE
MCI_ANIM_STEP_PARMS	Данная структура содержит информацию для установки положения в медиаданных для команды-сообщения MCI_STEP
MCI_ANIM_UPDATE_PARMS	Данная структура содержит информацию об обновлении размеров окна проигрывателя для команды-сообщения MCI_UPDATE
MCI_ANIM_WINDOW_PARMS	Данная структура содержит информацию для управления поведением окна проигрывателя для команды-сообщения MCI_ WINDOW

Таблица 3.8 *(продолжение)*

Структура	Описание
СТРУКТУРЫ ДЛЯ ЦИФРОВОГО ВИДЕОПРОИГРЫВАТЕЛЯ	
MCI_DGV_CAPTURE_PARMS	Данная структура предназначена для настройки захвата видеоданных для команды-сообщения MCI_CAPTURE
MCI_DGV_COPY_PARMS	Данная структура используется для настройки копирования сегмента медиаданных в буфер обмена и предназначена для команды-сообщения MCI_COPY. Также устанавливает, какие каналы (видео или аудио) копировать
MCI_DGV_CUE_PARMS	Данная структура предназначена для настройки позиции, с которой начнется воспроизведение или запись. Структура предназначена для команды-сообщения MCI_CUE
MCI_DGV_CUT_PARMS	Данная структура предназначена для настройки вырезания сегмента медиаданных в буфер обмена. Структура используется в команде-сообщении MCI_CUT. Также структура устанавливает, какие каналы (видео или аудио) нужно вырезать
MCI_DGV_DELETE_PARMS	Данная структура предназначена для настройки удаления сегмента аудиоданных. Используется для команды-сообщения MCI_DELETE
MCI_DGV_FREEZE_PARMS	Данная структура предназначена для команд-сообщений MCI_FREEZE и MCI_UNFREEZE
MCI_DGV_INFO_PARMS	Данная структура предназначена для получения общей информации о проигрывателе с помощью команды-сообщения MCI_INFO
MCI_DGV_LIST_PARMS	Данная структура используется для команды-сообщения MCI_LIST
MCI_DGV_LOAD_PARMS	Данная структура содержит информацию о загружаемом файле и используется в команде-сообщении MCI_LOAD
MCI_DGV_MONITOR_PARMS	Данная структура содержит информацию об источнике проигрываемых видеоданных и используется в команде-сообщении MCI_LOAD. Источник данных может быть в виде файла или внешнего сигнала, поступающего, например, через видеовход
MCI_DGV_OPEN_PARMS	Данная структура предназначена для команды-сообщения MCI_OPEN и содержит различные расширенные элементы. Если вы не хотите использовать расширенные элементы данной структуры, при работе с MCI_OPEN вы можете воспользоваться структурой MCI_Open_Parms

Таблица 3.8 *(продолжение)*

Структура	Описание
MCI_DGV_PASTE_PARMS	Данная структура используется для установки сегмента данных, вставляемых из буфера обмена, и предназначена для команды-сообщения MCI_PASTE. Также устанавливает, какие каналы (видео или аудио) нужно вставлять
MCI_DGV_PAUSE_PARMS	Структура предназначена для команды-сообщения MCI_PASTE. Данная структура аналогична структуре MCI_GENERIC_PARMS
MCI_DGV_PLAY_PARMS	Данная структура содержит информацию о начальном и конечном сегменте воспроизводимых медиаданных и предназначена для команды-сообщения MCI_PLAY. Если вы не хотите использовать расширенные элементы, вы можете воспользоваться структурой MCI_Play_Parms
MCI_DGV_PUT_PARMS	Данная структура содержит информацию о размерах изображения (размерах окна). Используется в команде-сообщении MCI_PUT
MCI_DGV_QUALITY_PARMS	Данная структура определяет уровень качества аудио- или видеоданных и используется в команде-сообщении MCI_QUALITY
MCI_DGV_RECORD_PARMS	Данная структура содержит информацию о начальной позиции записываемых медиаданных и предназначена для команды-сообщения MCI_ RECORD. Если вы не хотите использовать расширенные элементы, вы можете воспользоваться структурой MCI_RECORD_Parms
MCI_DGV_RECT_PARMS	Структура содержит информацию о размерах изображения (размерах окна). Используется в командах-сообщениях MCI_FREEZE, MCI_PUT, MCI_UNFREEZE и MCI_WHERE
MCI_DGV_RESERVE_PARMS	Структура содержит информацию о размерах зарезервированного проигрывателем пространства жесткого диска. Используется в команде-сообщении MCI_RESERVE
MCI_DGV_RESTORE_PARMS	Структура содержит информацию о графическом файле, bitmap которого вставляется в фрейм-буфер (frame buffer) проигрывателя. Используется в команде-сообщении MCI_RESTORE
MCI_DGV_RESUME_PARMS	Структура используется для команды-сообщения MCI_RESUME. Данная структура аналогична структуре MCI_GENERIC_PARMS

Таблица 3.8 *(продолжение)*

Структура	Описание
MCI_DGV_SAVE_PARMS	Данная структура содержит расширенную информацию о сохраняемом файле и размерах изображения (окна). Используется в команде-сообщении MCI_Save. Если вы не хотите использовать расширенные элементы, вы можете воспользоваться структурой MCI_SAVE_PARMS
MCI_DGV_SET_PARMS	Структура содержит данные, настраивающие опциональное состояние проигрывателя. Используется в команде-сообщении MCI_SET
MCI_DGV_SETAUDIO_PARMS	Структура содержит информацию об аудиокомпрессоре (кодеке) звуковой дорожки, а также об уровне и формате аудиосигнала. Данная структура предназначена для настройки аудиосигнала при проигрывании и захвате. Используется в команде-сообщении MCI_SETAUDIO
MCI_DGV_SETVIDEO_PARMS	Структура содержит информацию о видеокомпрессоре (кодеке) видеодорожки. Данная структура предназначена для настройки видеосигнала при проигрывании и захвате. Используется в команде-сообщении MCI_SETVIDEO
MCI_DGV_SIGNAL_PARMS	Данная структура содержит информацию о маркерах (указателях) позиций в видеофайле. Используется в команде-сообщении MCI_SIGNAL
MCI_DGV_STATUS_PARMS	Данная структура предназначена для получения статусной информации о проигрывателе. Используется в команде-сообщении MCI_STATUS
MCI_DGV_STEP_PARMS	Данная структура содержит информацию о фрейме, который будет выбран (позиционирован) с помощью команды-сообщения MCI_STEP
MCI_DGV_STOP_PARMS	Структура используется для команды-сообщения MCI_STOP. Данная структура аналогична структуре MCI_GENERIC_PARMS
MCI_DGV_UPDATE_PARMS	Данная структура содержит информацию об окне проигрывателя, содержимое которого будет обновлено с помощью команды-сообщения MCI_UPDATE
MCI_DGV_WINDOW_PARMS	Структура используется для команды-сообщения MCI_WINDOW. Данная структура аналогична структуре MCI_GENERIC_PARMS

Таблица 3.8 *(продолжение)*

Структура	Описание
СТРУКТУРЫ ДЛЯ ВИДЕОМАГНИТОФОНА	
MCI_VCR_CUE_PARMS	Данная структура предназначена для настройки позиции, с которой начнется воспроизведение или запись. Структура предназначена для команды-сообщения MCI_CUE
MCI_VCR_LIST_PARMS	Данная структура используется для команды-сообщения MCI_LIST
MCI_VCR_PLAY_PARMS	Данная структура содержит информацию о начальном и конечном сегменте воспроизводимых медиаданных и предназначена для команды-сообщения MCI_PLAY. Если не хотите использовать расширенные элементы, можете воспользоваться структурой MCI_PLAY_Parms
MCI_VCR_RECORD_PARMS	Данная структура содержит информацию о начальной позиции записываемых медиаданных и предназначена для команды-сообщения MCI_ RECORD. Если не хотите использовать расширенные элементы, можете воспользоваться структурой MCI_RECORD_Parms
MCI_VCR_SEEK_PARMS	Данная структура содержит информацию для позиционирования. Используется в команде-сообщении MCI_SEEK
MCI_VCR_SET_PARMS	Структура содержит данные, настраивающие опциональное состояние видеопроигрывателя. Используется в команде-сообщении MCI_SET
MCI_VCR_SETAUDIO_PARMS	Структура содержит информацию для команды-сообщения MCI_SETAUDIO
MCI_VCR_SETVIDEO_PARMS	Структура содержит информацию для команды-сообщения MCI_SETVIDEO
MCI_VCR_SETTUNER_PARMS	Структура содержит информацию об устанавливаемом канале на тюнере видеомагнитофона. Структура используется в команде-сообщении MCI_SETTUNER
MCI_VCR_STATUS_PARMS	Данная структура предназначена для получения статусной информации о состоянии видеомагнитофона. Используется в команде-сообщении MCI_STATUS
MCI_VCR_STEP_PARMS	Данная структура содержит информацию о позиции на ленте, которая будет выбрана (позиционирована) с помощью команды-сообщения MCI_STEP

Таблица 3.8 *(окончание)*

Структура	Описание
СТРУКТУРЫ ДЛЯ ПРОИГРЫВАТЕЛЯ ВИДЕОДИСКОВ	
MCI_VD_ESCAPE_PARMS	Данная структура содержит Escape-строку, отправляемую проигрывателю в качестве команды. Используется в команде-сообщении MCI_ESCAPE
MCI_VD_PLAY_PARMS	Данная структура содержит информацию о начальном и конечном сегментах воспроизводимых медиаданных и предназначена для команды-сообщения MCI_PLAY. Если не хотите использовать расширенные элементы, вы можете воспользоваться структурой MCI_PLAY_Parms
MCI_VD_STEP_PARMS	Данная структура содержит информацию о фрейме, который будет выбран (позиционирован) с помощью команды-сообщения MCI_STEP
СТРУКТУРЫ ДЛЯ ВИДЕОМАГНИТОФОНА	
MCI_WAVE_DELETE_PARMS	Данная структура содержит информацию о начальном и конечном сегменте удаляемой звуковой дорожки. Структура используется командой-сообщением MCI_DELETE
MCI_WAVE_OPEN_PARMS	Данная расширенная структура используется для команды-сообщения MCI_OPEN
MCI_WAVE_SET_PARMS	Структура содержит данные, настраивающие опциональное состояние wave-проигрывателя. Используется в команде-сообщении MCI_SET

Еще одной особенностью MCI-структур является нестандартное использование TRect-структур. В MCI TRect-структуры обрабатываются не так, как обычно. Различие состоит в том, что в MCI используются только два поля — rc.right и rc.bottom, в которых содержится информация о ширине и высоте прямоугольника. Следует это иметь в виду при разработке приложений.

Практика использования

После изучения теоретического материала по MCI приступим к практическим вопросам. Самым первым нашим проектом в изучении MCI будет приложение, предназначенное для изучения интерфейса командных строк MCI (как самого простого) и отладки последовательностей команд. Внешний вид приложения вы можете увидеть на рис. 3.2.

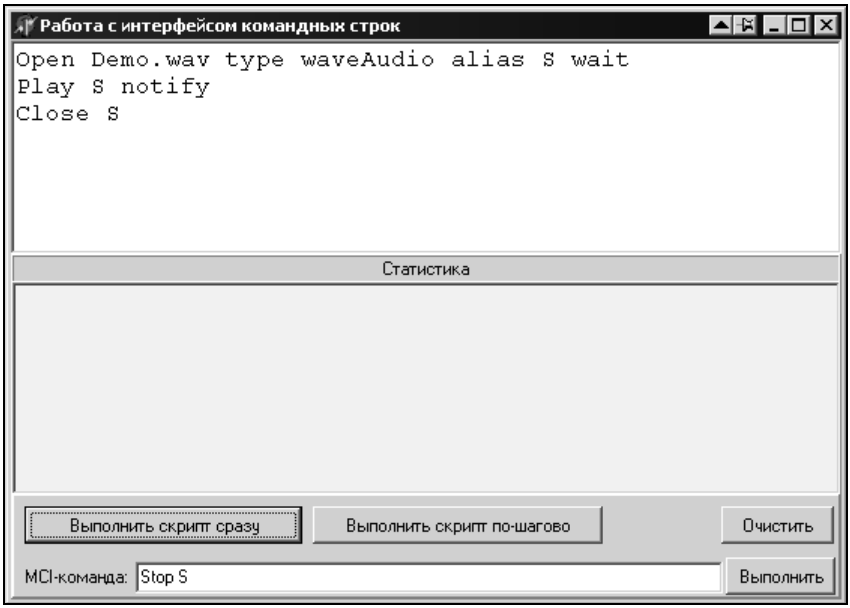


Рис. 3.2. Внешний вид приложения CommandStringsMCI

Как видно на приведенном рисунке, окно, расположенное в верхней части экрана, предназначено для ввода командных строк. Это окно представляет собой многострочный редактор текста, поэтому вы можете вводить сразу несколько строк команд. Введенные в многострочном редакторе команды можно выполнять по отдельности, нажимая кнопку **Выполнить скрипт пошагово**, или все вместе, нажимая кнопку **Выполнить скрипт сразу**. Также вы можете запускать MCI-команды из командной строки с помощью кнопки **Выполнить**. В окне **Статистика** отображается результат выполнения данных MCI-команд. Если во время выполнения произошла какая-либо ошибка, ее текстовое описание появится в данном окне. Также если в качестве одного из параметров команды был указан флаг `notyfy`, в окне **Статистика** будет отображен результат, переданный с сообщением `MM_MCINOTIFY`.

Исходный код данной программы вы сможете найти на прилагаемом компакт-диске в каталоге Source\Ch03\Ex01.

Проигрывание wave-файлов

Наверно рано или поздно любой программист, работающий с мультимедиа, захочет написать свой проигрыватель. В этом разделе мы покажем, что с помощью MCI несложно создать небольшой и простой проигрыватель wave-файлов. Конечно, мы не ставим задачу написать что-то вроде WinAmp или

JetAudio — наша цель показать основные возможности использования MCI в приложениях. Внешний вид проигрывателя представлен на рис 3.3.

Рис. 3.3. Простой wave-проигрыватель

Хотя интерфейс приложения не выглядит функциональным, приложение вполне работоспособно и при желании вы легко можете его модифицировать до ваших нужд. В листинге 3.3 представлен краткий исходный код нашего проигрывателя, полный исходный код его находится на прилагаемом компакт-диске в каталоге Source\Ch03\Ex02.

Листинг 3.3. Простой проигрыватель wave-файлов

```
var
  FormMain: TFormMain;
  // Основные используемые MCI-структуры
  MciOpenParms  : TMCI_Wave_Open_Parms;
  MciPlayParm   : TMCI_Play_Parms;
  MciGenericParm: TMCI_Generic_Parms;
  MciSeekParm   : TMCI_Seek_Parms;
  MciSetParm    : TMCI_Set_Parms;
  MciStatusParm : TMCI_Status_Parms;
  // Флаги для команд
  Flags   : DWORD;

implementation

{$R *.dfm}

// Открыть MCI-устройство и загрузить файл
procedure TFormMain.OpenFile(FileName: String);
begin
  // Открываем устройство MCI_DEVTYPE_WAVEFORM_AUDIO
  mciOpenParms.lpstrDeviceType := PChar(MCI_DEVTYPE_WAVEFORM_AUDIO);
  mciOpenParms.lpstrElementName:= PChar(FileName);
  Flags := 0;
  Flags := MCI_OPEN_TYPE or MCI_OPEN_ELEMENT or MCI_OPEN_TYPE_ID;
```

```
  ErrorID := mciSendCommand(0, MCI_OPEN, Flags, Longint(@mciOpenParms));
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  // Получаем идентификатор устройства
  dwDeviceID := mciOpenParms.wDeviceID;
  // Устанавливаем формат времени ------------------------------------
  MciSetParm.dwCallback := 0;
  MciSetParm.dwTimeFormat := mci_Format_Samples;
  Flags := 0;
  Flags := mci_Set_Time_Format;
  ErrorID := mciSendCommand(dwDeviceID, mci_Set, Flags,
                            Longint(@MciSetParm));
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  // Получаем длину проигрываемого файла в семплах-------------------
  MciStatusParm.dwCallback := 0;
  // Получить продолжительность
  MciStatusParm.dwItem     := mci_Status_Length;
  Flags := 0;
  Flags := mci_Status_Item; // Получить статус в MciStatusParm.dwReturn
  ErrorID := mciSendCommand(dwDeviceID, mci_Status, Flags ,
                            Longint(@MciStatusParm));
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  // Получить длину
  WaveLength := MciStatusParm.dwReturn;
  ProgressBar.Min := 0;
  ProgressBar.Max := WaveLength;
  ProgressBar.Position := ProgressBar.Min;
  StatusBar.SimpleText := FileName;
  FlgOpen      := True;
end;

// Проиграть с начала
procedure TFormMain.BitBtnPlayClick(Sender: TObject);
begin
  if dwDeviceID = 0 then
     Exit;
```

```
  // Встаем на начало ------------------------------------------------
  MciSeekParm.dwCallback:= 0;  // Не устанавливаем хендл окна, т. к. не
                               // собираемся получать
                               // сообщения от данной команды
  Flags := 0;
  Flags := MCI_SEEK_TO_START;
  ErrorID := mciSendCommand(dwDeviceID ,mci_Seek, Flags,
                            Longint(@MciSeekParm));
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  // Запуск на проигрывание ------------------------------------------
  MciPlayParm.dwCallback := Handle;
  Flags := 0;
  Flags := MCI_NOTIFY;  // Подавать сообщение о выполнении команды +
                        // проигрывать в фоне
  ErrorID := mciSendCommand(dwDeviceID ,mci_Play, Flags,
                            Longint(@MciPlayParm));
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  ProgressBar.Position := ProgressBar.Min;
  Timer.Enabled := True;
end;

// Закрыть
procedure TFormMain.CloseMCIDevice;
begin
  MciGenericParm.dwCallback := Handle;
  Flags := 0;
  ErrorID := mciSendCommand(dwDeviceID ,MCI_Close, Flags,
             Longint(@MciGenericParm));
end;

// Стоп
procedure TFormMain.BitBtn3Click(Sender: TObject);
begin
  if dwDeviceID = 0 then
     Exit;
  MciGenericParm.dwCallback := Handle;
  Flags := 0;
```

```
  ErrorID := mciSendCommand(dwDeviceID ,MCI_STOP, Flags,
             Longint(@MciGenericParm));
end;

// ОПРЕДЕЛЕНИЕ ТЕКУЩЕЙ ПОЗИЦИИ В МЕДИАДАННЫХ
procedure TFormMain.TimerTimer(Sender: TObject);
begin
  // Получаем позицию в медиаданных
  MciStatusParm.dwCallback := 0;
  MciStatusParm.dwItem     := mci_Status_Position;
  Flags := 0;
  Flags := mci_Status_Item; // Получить статус в MciStatusParm.dwReturn
  ErrorID := mciSendCommand(dwDeviceID, mci_Status, Flags ,
             Longint(@MciStatusParm));
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  // Получить длину и отобразить в прогресс-баре
  ProgressBar.Position := MciStatusParm.dwReturn;
  ProgressBar.Repaint;
end;

// ПЕРЕХВАТ СООБЩЕНИЯ О ЗАВЕРШЕНИИ MCI-команды
procedure TFormMain.MMNotify(var Message: TMessage);
begin
  case Message.WParam of
    mci_Notify_Successful: ShowMessage('Проиграно успешно');
    mci_Notify_Superseded: ShowMessage('Прервано другой командой');
    mci_Notify_Aborted   : ShowMessage('Прервано пользователем');
    mci_Notify_Failure   : ShowMessage('Ошибка');
  end;
  Timer.Enabled := False;
end;

// ПОКАЗ ОШИБОК
procedure TFormMain.ShowMCIError;
begin
  mciGetErrorString(ErrorID, ErrMsg, SizeOf(ErrMsg));
  SetString(ErrMsgStr, ErrMsg, StrLen(ErrMsg));
  ShowMessage('Ошибка MCI!' + #10#13 +  ErrMsgStr);
end;
```

Проигрывание MIDI-файлов

В данном разделе мы рассмотрим создание проигрывателя для MIDI-файлов. Прежде чем мы остановимся на программной реализации проигрывателя, рассмотрим, что представляют собой файлы формата MIDI.

Стандарт MIDI (Musical Instrument Digital Interface — цифровой интерфейс музыкальных инструментов) был разработан для совместимости различных электронных музыкальных инструментов (синтезаторов, музыкальных клавиатур) с компьютером, а также друг с другом. Можно сказать, что формат MIDI хранит последовательность команд, предназначенных для управления различными инструментами через различные каналы.

MCI поддерживает так называемый авторизованный формат MIDI, представленный в файлах с расширение mid. Следует отметить, что данный формат не совсем соответствует стандарту RIFF. Файлы же, полностью соответствующие стандарту, имеют расширение rmi, однако MCI не поддерживаются.

Данное различие объясняется тем, что в спецификации MIDI определены 16 логических каналов, предназначенных для адресации 16 логических синтезаторов. Данные каналы по стандарту распределены следующим образом: каналы с номерами 13...16 используются синтезаторами базового уровня, каналы с номерами 1...10 — синтезаторами расширенного уровня.

Синтезатор базового уровня содержит в себе голоса трех мелодичных и трех ударных инструментов. Уровень полифонии такого синтезатора равен 6 для мелодичных и 3 — для ударных инструментов. Это означает, что синтезатор базового уровня может играть одновременно 6 нот на мелодичных инструментах и 3 ноты на ударных. Синтезаторы расширенного уровня содержат в себе 9 мелодичных и 8 ударных инструментов при уровне полифонии, равном 16.

Однако не все музыкальные синтезаторы имеют одинаковое распределение каналов и инструментов, вследствие этого и появились два формата MIDI.

После краткого обзора остановимся на нашем примере. Как вы уже заметили, первый пример проигрывателя wave-файлов не отличался особой функциональностью, в частности в нем нельзя было сделать перемотку на заданную позицию. Поэтому в данном примере мы постарались учесть некоторые недостатки. В частности, мы вам покажем приемы, позволяющие работать с позицией в медиаданных, а также реализуем интересную особенность MIDI — возможность задания темпа проигрывания.

Для реализации позиционирования на выбранное место мы немного изменили процедуру, включающую воспроизведение, сокращенный код процедуры представлен в листинге 3.4.

Листинг 3.4. Реализация воспроизведения с заданной позиции

```
procedure TFormMain.Play(Position: Cardinal);
begin
MciPlayParm.dwCallback := Handle;
MciPlayParm.dwFrom := Position;
Flags := MCI_NOTIFY or MCI_FROM ;
ErrorID := mciSendCommand(dwDeviceID, MCI_PLAY, Flags,
             Longint(@MciPlayParm));
end;
```

В данной процедуре появилась возможность указания начальной позиции из переменной `Position`. Достигается это за счет использования флага `MCI_FROM` и указания позиции в поле `dwFrom` структуры `MciPlayParm`. Далее мы положили на форму `ScrollBar` и в обработчике события `ScrollBarScroll` прописали следующий код, представленный в листинге 3.5.

Листинг 3.5. Обработка перемещения бегунка

```
procedure TFormMain.ScrollBarScroll(Sender: TObject;
    ScrollCode: TScrollCode; var ScrollPos: Integer);
var
Prom: Integer;
begin
// Конец перетаскивания бегунка
IF ScrollCode = scEndScroll then
Begin
  // Если проигрыватель играет, то продолжить игру с позиции
  // бегунка, иначе —
 // позиционирование на выбранное время
  IF IsPlay then
  begin
   Play(ScrollBar.Position)
  end
  else
  begin
   MciSeekParm.dwCallback := 0;
   MciSeekParm.dwTo := ScrollBar.Position;
   Flags := 0;
   Flags := MCI_TO;
   ErrorID := mciSendCommand(dwDeviceID, mci_SEEK, Flags,
         Longint(@MciSeekParm));
 end;
end
```

```
else
 Timer.Enabled := False;
end;
```

Смысл данного кода прост: если пользователь перемещает бегунок ScrollBar, то в зависимости от режима работы MCI-устройства Sequencer проигрыватель либо перескакивает на проигрывание с заданного места и продолжает проигрывание, либо просто позиционируется на заданное место. Функция `IsPlay`, представленная в коде, получает опциональное состояние режима устройства и, если устройство находится в режиме воспроизведения, возвращает `TRUE`. Функция `IsPlay` представлена в листинге 3.6.

Листинг 3.6. Получение опционального состояния устройства

```
function TFormMain.IsPlay: Boolean;
begin
  Result := False;
  MciStatusParm.dwCallback := 0;
  MciStatusParm.dwItem     := MCI_STATUS_MODE;
  Flags := 0;
  Flags := mci_Status_Item;   // Получить режим в MciStatusParm.dwReturn
  ErrorID := mciSendCommand(dwDeviceID, mci_Status, Flags ,
                            Longint(@MciStatusParm));
  Result := MciStatusParm.dwReturn = MCI_MODE_PLAY ;
end;
```

Для реализации возможности манипулирования темпом проигрывания MIDI нами были созданы следующие процедуры и функции, сокращенный код которых представлен в листинге 3.7.

Листинг 3.7. Функции управления темпом проигрывания

```
//---------------------------------------------------------------------
// Устанавливаем скорость проигрывания MIDI (темп)
//---------------------------------------------------------------------
procedure TFormMain.SetTempo(Value: Cardinal);
begin
  MciSetSeqParm.dwCallback   := 0;
  MciSetSeqParm.dwTempo      := Value;
  Flags := 0;
  Flags := MCI_SEQ_SET_TEMPO;
  ErrorID := mciSendCommand(dwDeviceID, mci_Set, Flags ,
                            Longint(@MciSetSeqParm));
end;
```

```
//---------------------------------------------------------------------------
// Получаем скорость проигрывания MIDI (темп)
//---------------------------------------------------------------------------
function TFormMain.GetTempo: Cardinal;
begin
  MciStatusParm.dwCallback := 0;
  MciStatusParm.dwItem     := MCI_SEQ_STATUS_TEMPO;
  Flags := 0;
  Flags := MCI_STATUS_ITEM or MCI_WAIT;
  ErrorID := mciSendCommand(dwDeviceID, MCI_STATUS, Flags,
             Longint(@MciStatusParm));
  Result := MciStatusParm.dwReturn;
end;
```

Полный исходный код приложения представлен на прилагаемом компакт-диске в каталоге Source\Ch03\Ex03.

Далее мы остановимся на некоторых особенностях, присущих MCI-устройству SEQUENCER.

- SEQUENCER не поддерживает следующие аргументы командной строки `SET`:
  `audio all off`, `audio all on`, `audio left on`, `audio left off`, `audio right on`, `audio right off`.
- SEQUENCER имеет дополнительные аргументы командной строки `SET`, такие как: `time format song pointer` — установка формата времени в единицах 1/16 ноты, а также `tempo`, `port`, `offset`, `master`, `slave`.

Звукозапись

После того как мы с вами создали два различных проигрывателя, перейдем к звукозаписи. Для демонстрации возможностей MCI по записи мы создали небольшое демонстрационное приложение, внешний вид которого вы можете увидеть на рис. 3.4.

Данное приложение очень похоже на приложение "Звукозапись" в стандартной поставке Windows, хотя и менее функционально. Перед записью вы можете выставить различные настройки записываемого сигнала, такие как количество каналов, дискретность сигнала и частоту дискретизации. Исходный код данного приложения представлен на прилагаемом компакт-диске в каталоге Source\Ch03\Ex04. Наиболее интересной функцией в плане изучения MCI служит функция, подготавливающая MCI-устройство к записи и устанавливающая качество записи. Данная функция сокращенно представлена в листинге 3.8.

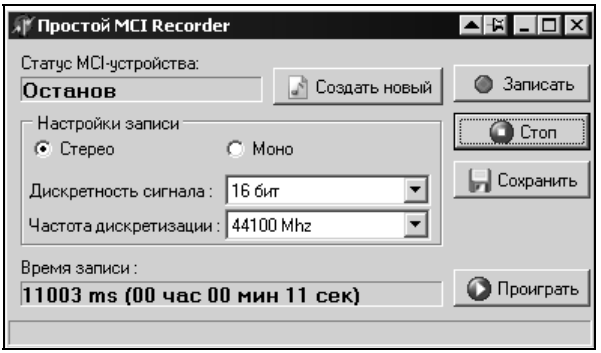


Рис. 3.4. Приложение для звукозаписи

Листинг 3.8. Подготовка MCI-устройства к записи

```
var
  // Основные используемые MCI-структуры
  MciWaveSetParm: TMCI_Wave_Set_Parms;

procedure TFormMain.BitBtnRecordClick(Sender: TObject);
var
  SamplesPerSec,BPS : Integer;
begin
  if dwDeviceID = 0 then
     Exit;
  // Устанавливаем опциональные настройки устройства --------------------
  MciWaveSetParm.dwCallback   := 0;
  // формат времени - в миллисекундах
  MciWaveSetParm.dwTimeFormat := MCI_FORMAT_MILLISECONDS ;
  // Устанавливаем кол-во каналов - моно/стерео
  IF RadioButtonMONO.Checked then
     MciWaveSetParm.nChannels    := 1
  else
     MciWaveSetParm.nChannels    := 2;
  // Устанавливаем частоту дискретизации и кол-во бит
  Case ComboBox.ItemIndex of
       0 : SamplesPerSec := 44100;
       1 : SamplesPerSec := 22050;
       2 : SamplesPerSec := 11025;
  end;
  // В поле nSamplesPerSec записана частота дискретизации, т. е.
  // количество выборок сигнала в секунду.
```

```
// В этом поле могут находиться стандартные
// значения 11025 кГц, 22050 кГц или 44100 кГц
MciWaveSetParm.nSamplesPerSec  := SamplesPerSec;
// Дискретность сигнала  - Обычно используются значения 8 или 16.
Case ComboBoxDiskret.ItemIndex of
     0 : MciWaveSetParm.wBitsPerSample  := 16;
     1 : MciWaveSetParm.wBitsPerSample  := 8;
end;
// Поле nAvgBytesPerSec содержит среднюю скорость потока данных,
// т. е. количество байт в секунду, передаваемое драйверу устройства
// Для монофонического сигнала с дискретностью 8 бит численное значение
// скорости совпадает со значением частоты дискретизации.
// Для стереофонического сигнала с дискретностью 8 бит она в два раза
// выше.
MciWaveSetParm.nAvgBytesPerSec := Round((MciWaveSetParm.nChannels *
                                  SamplesPerSec *
                                  MciWaveSetParm.wBitsPerSample) / 8);
// В поле nBlockAlign находится выравнивание блока в байтах
MciWaveSetParm.nBlockAlign    :=  Round((MciWaveSetParm.nChannels *
                                  MciWaveSetParm.wBitsPerSample) / 8);

Flags := 0;
Flags := MCI_SET_TIME_FORMAT or        // установить время
         MCI_WAVE_SET_CHANNELS or      // установить каналы
         MCI_WAVE_SET_BITSPERSAMPLE or // установить частоту выборок
         MCI_WAVE_SET_BITSPERSAMPLE or // установить выравнивание блока
         MCI_WAVE_SET_SAMPLESPERSEC;
ErrorID := mciSendCommand(dwDeviceID, mci_Set, Flags ,
           Longint(@MciWaveSetParm));
IF ErrorID <> 0 then
begin
   ShowMCIError; Exit;
end;

MciRecordParm.dwCallback := Handle;
MciRecordParm.dwFrom := 0;
MciRecordParm.dwTo   := 10000;
Flags := 0;
Flags := MCI_NOTIFY;  // Подавать сообщение о выполнении команды
ErrorID := mciSendCommand(dwDeviceID ,mci_Record, Flags,
           Longint(@MciRecordParm));
IF ErrorID <> 0 then
begin
```

```
    ShowMCIError; Exit;
  end;
end;
```

Проигрывание Audio-CD

Следующим нашим примером, иллюстрирующим работу с MCI-устройствами, стало приложение, позволяющее проигрывать Audio-CD, внешний вид которого вы можете рассмотреть на рис. 3.5.

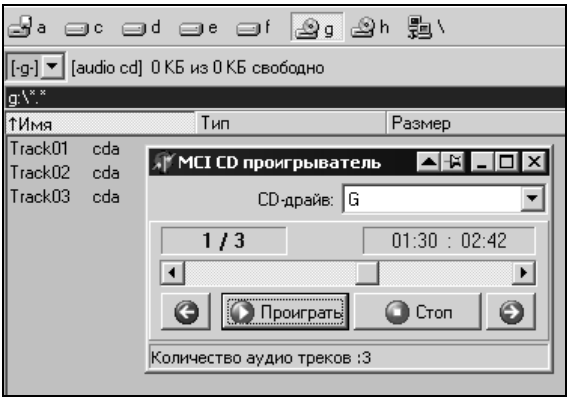


Рис. 3.5. Простой проигрыватель Audio-CD

Полный исходный код приложения представлен на прилагаемом компакт-диске в каталоге Source\Ch03\Ex05. Здесь же мы остановимся на некоторых особенностях, присущих MCI-устройству CDAUDIO, которые обязан знать программист.

- *Проигрывание* — по окончании проигрывания заданного трека и при наличии следующих треков CDAUDIO автоматически переключится на следующий трек, поэтому вы должны предусмотреть в своей программе отслеживание смены трека, если в программе предусмотрен какой-либо счетчик положения. Сделать это несложно при использовании таймера и MCI-команды `Status`.
- *Определение текущей позиции* — здесь скрыта небольшая хитрость. Дело в том, что текущая позиция в треке возвращается в формате TMSF (трек / минута / секунда / фрейм), а длина самого трека — в формате MSF (минута/секунда / фрейм). Это объясняется тем, что при определении длины трека в качестве параметров MCI-команды `STATUS` используется соответствующий номер трека, проще говоря, вы спрашиваете у устройства, какую длину имеет трек № N, поэтому MCI и возвращает информацию в MSF, т. к. номер трека уже известен. Другое дело — определение текущей пози-

ции. При запросе вы можете и не знать, на каком конкретно треке позиционировано устройство CDAUDIO в данный момент времени, поэтому MCI и возвращает информацию в TMSF. Поэтому, если вы хотите сделать правильное отображение текущей позиции в треке, вы должны это учитывать.

- *Проверка трека* — перед работой с конкретным треком вы должны его проверить, является ли данный трек на компакт-диске звуковой дорожкой, и если является, то затем производите необходимые действия. Данная проверка обусловлена тем, что на компакт-диске помимо аудиотреков могут быть записаны треки с данными. Такие диски очень часто используются в компьютерных играх, когда в звуковых треках записана фоновая музыка игры, а в треках данных — сама игра.

Проигрывание видеофайлов AVI

Последним приложением, иллюстрирующим работу с MCI, стал пример простого проигрывателя видеофайлов. В данном приложении для управления MCI-устройством вместо интерфейса команд-сообщений MCI мы использовали интерфейс командных строк. Код, демонстрирующий использование командных строк, в сокращенном виде вы можете увидеть в листинге 3.9. Полный исходный код представлен на прилагаемом компакт-диске в каталоге Source\Ch03\Ex06.

Листинг 3.9. Использование командных строк на примере простого проигрывателя видеофайлов

```
// Закрываем устройство
procedure TFormMain.CloseMCIDevice;
begin
  Stop; // Остановить на всякий случай
  // Закрыть
  CommandStr   := Format('Close video',[]);
  ErrorID := mciSendString(PChar(CommandStr),nil, 0, 0);
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  OnOff(False);
end;

// Открыть MCI-устройство и загрузить файл
procedure TFormMain.OpenFile(FileName: String);
var
  Coord : TSepRec;
```

```
begin
  // Загружаем файл
  CommandStr   := Format('Open "%S" type avivideo alias video Parent %D
                         style child',[FileName, Window.Handle]);
  ErrorID := mciSendString(PChar(CommandStr),nil, 0, 0);
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  // Устанавливаем формат времени ----------------------------------------
  CommandStr   := Format('Set video time format milliseconds',[]);
  ErrorID := mciSendString(PChar(CommandStr),nil, 0, 0);
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  // Получаем длину проигрываемого файла в -------------------------------
  CommandStr   := Format('Status video length',[]);
  ErrorID := mciSendString(PChar(CommandStr),ReturnStr,
                           Length(ReturnStr), 0);
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  SetString(Ret, ReturnStr, StrLen(ReturnStr));

  // Получить длину
  AVILength := StrToInt(Ret);
  ScrollBar.Min := 0;
  ScrollBar.Max := AVILength;
  ScrollBar.Position := ScrollBar.Min;

  StaticTextTime.Caption := MillisecondsToTimeStr(AVILength);
  StatusBar.SimpleText := FileName;
  FlgOpen      := True;

  // Определяем высоту и ширину выводимого изображения
  CommandStr   := Format('Where video destination',[]);
  ErrorID := mciSendString(PChar(CommandStr),ReturnStr,
             Length(ReturnStr), 0);
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  SetString(Ret, ReturnStr, StrLen(ReturnStr));
```

```
  // Разбиваем строку на элементы
  Coord := GetSeparatorRec(Ret,' ');
  // Настраиваем окно проигрывателя
  Window.Width  := StrToInt(Coord.Rec[3]) + 7;  // поправки
  Window.Height := StrToInt(Coord.Rec[4]) + 24;
  Window.Left   := (Left + (Width div 2)) - (Window.Width Div 2);
  Window.Top    := top - Window.Height;
  Window.Show;
  // Обновляем окно видео
  CommandStr   := Format('put video window at %S',[Ret]);
  ErrorID := mciSendString(PChar(CommandStr),nil, 0, 0);
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
end;

// Проиграть
procedure TFormMain.Play(Position: Cardinal);
begin
 CommandStr   := Format('Play video from %D',[Position]);
 ErrorID := mciSendString(PChar(CommandStr),nil, 0, 0);
 IF ErrorID <> 0 then
 begin
    ShowMCIError; Exit;
 end;
 Timer.Enabled:= True;
end;

// Стоп
procedure TFormMain.Stop;
begin
 CommandStr   := Format('Stop video',[]);
 ErrorID := mciSendString(PChar(CommandStr),nil, 0, 0);
 Timer.Enabled := False;
end;

// ОПРЕДЕЛЕНИЕ ТЕКУЩЕЙ ПОЗИЦИИ В МЕДИАДАННЫХ
procedure TFormMain.TimerTimer(Sender: TObject);
begin
  // Получаем текущую позицию в медиаданных
  CommandStr   := Format('Status video Position',[]);
```

```
  ErrorID := mciSendString(PChar(CommandStr),ReturnStr,
                           Length(ReturnStr), 0);
  IF ErrorID <> 0 then
  begin
     ShowMCIError; Exit;
  end;
  SetString(Ret, ReturnStr, StrLen(ReturnStr));
  ScrollBar.Position := StrToInt(Ret);
  StaticTextPos.Caption := MillisecondsToTimeStr(ScrollBar.Position);
end;
```

Внешний вид приложения вы можете рассмотреть на рис. 3.6.

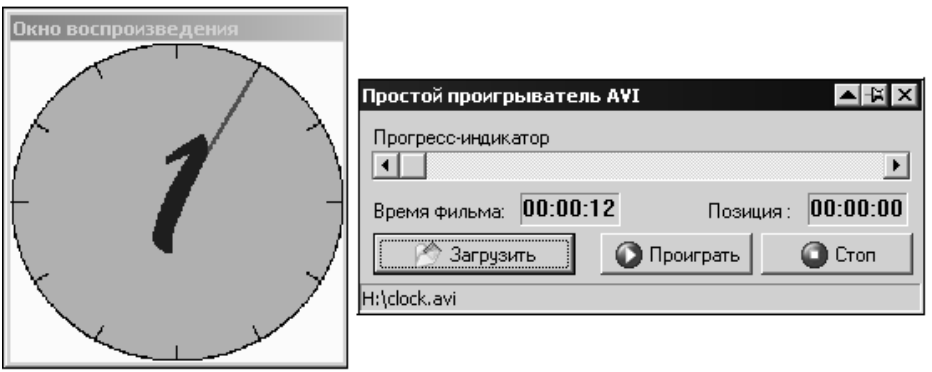


Рис. 3.6. Простой проигрыватель AVI-файлов

Коды ошибок MCI

В табл. 3.9 мы постарались привести описания ошибок в соответствии с их кодами и дать советы по возможному их разрешению.

Таблица 3.9. *Ошибки MCI*

Код ошибки	Описание и способ устранения
MCIERR_BAD_TIME_FORMAT	Неправильный формат времени. Данная ошибка часто появляется при установке формата времени в MCI-команде SET для заданного устройства и вызвана тем, что данное MCI-устройство не может работать с таким форматом времени. Установить корректный формат времени

Таблица 3.9 *(продолжение)*

Код ошибки	Описание и способ устранения
MCIERR_CANNOT_LOAD_DRIVER	Невозможно загрузить драйвер. Данная ошибка может быть вызвана при открытии MCI-устройства. Ошибка свидетельствует об отсутствии необходимого драйвера для идентификатора MCI-устройства. Это довольно серьезная ошибка может быть также связана с конфликтами устройств в системе
MCIERR_CANNOT_USE_ALL	Для этой команды нельзя использовать строку `all` в качестве имени устройства. Ошибка связана с неверным синтаксисом команды
MCIERR_CREATEWINDOW	Невозможно создать или использовать окно. Данная ошибка проявляется при неверно заданном хендле окна. Например, при разработке приложения, связанного с проигрыванием цифровых видеофильмов, вы неправильно указали хэндл окна, на которое будет происходить вывод изображения средствами MCI
MCIERR_DEVICE_LENGTH MCIERR_DEVICE_ORD_LENGTH	Превышено ограничение на длину имени устройства или драйвера (больше 79 символов). Данная ошибка может проявиться в командных строках MCI. При использовании команд-сообщений вероятность вызова данной ошибки сводится к нулю
MCIERR_DEVICE_LOCKED	Устройство закрыто или временно заблокировано, поэтому не может выполнить MCI-команду. Повторите команду через небольшой промежуток времени
MCIERR_DEVICE_NOT_INSTALLED MCIERR_DEVICE_TYPE_REQUIRED	Указанное устройство не установлено или не найдено в системе. Установите необходимое ПО или драйверы. Часто возникает при использовании программных MCI-устройств, например, цифровых проигрывателей
MCIERR_DEVICE_NOT_READY	Устройство MCI не готово в данный момент времени выполнить вашу команду
MCIERR_DEVICE_OPEN	Ошибка при открытии MCI-устройства. Возникает, если устройство уже открыто. Закройте устройство
MCIERR_DRIVER MCIERR_DRIVER_INTERNAL	Ошибка драйвера MCI-устройства или конфликт версий драйверов. Следует установить драйвер более поздней версии

Таблица 3.9 *(продолжение)*

Код ошибки	Описание и способ устранения
MCIERR_DUPLICATE_ALIAS	Указанный алиас MCI-устройства уже используется в системе
MCIERR_EXTENSION_NOT_FOUND	Указанному расширению имени файла не соответствует ни одно MCI-устройство, установленное в системе. Данная ошибка возникает при открытии устройства, автоматически связанного с расширением файла. Проверьте, установлено ли данное MCI-устройство в системе, или укажите его явно
MCIERR_EXTRA_CHARACTERS	Текстовая строка должна быть заключена в двойные кавычки. Если появилась эта ошибка, в строке есть символы, расположенные после закрывающей кавычки
MCIERR_FILE_NOT_FOUND	Файл не найден
MCIERR_FILE_NOT_SAVED MCIERR_FILE_WRITE	Ошибка, проявляющаяся при сохранении файла с помощью MCI-команды `Save`. Ошибка часто вызвана нехваткой свободного дискового пространства
MCIERR_FILE_READ	Ошибка при чтении файла. Ошибка может быть вызвана неправильно сопоставленным форматом файла с конкретным MCI-устройством
MCIERR_FLAGS_NOT_COMPATIBLE	Указанные во флагах команды параметра не совместимы. Проверьте флаги на совместимость между собой и MCI-устройством
MCIERR_FILENAME_REQUIRED	Требуется указать правильное имя файла
MCIERR_GET_CD	Не найден нужный файл или драйвер
MCIERR_HARDWARE	Ошибка аппаратного обеспечения
MCIERR_ILLEGAL_FOR_AUTO_OPEN	Данную команду нельзя выполнить для устройства, открываемого автоматически
MCIERR_INTERNAL	Ошибка при инициализации MCI. Необходимо перезапустить операционную систему
MCIERR_INVALID_DEVICE_ID	Неверный идентификатор MCI-устройства
MCIERR_INVALID_DEVICE_NAME	Неправильное имя устройства. MCI не может открыть устройство, определенное данным именем

Таблица 3.9 *(продолжение)*

Код ошибки	Описание и способ устранения
MCIERR_INVALID_FILE	Файл, открытый MCI-устройством, невозможно проиграть, т. к. данное устройство не поддерживает его формат. Ошибка может возникнуть, например, когда вы пытаетесь проиграть файл MP3 устройством waveaudio
MCIERR_INVALID_SETUP	Неправильная установка параметров MIDI. Необходимо восстановить оригинальный файл midimap.cfg, расположенный в системном каталоге Windows
MCIERR_MISSING_INTEGER MCIERR_NO_INTEGER	Команда ожидает необходимый параметр в виде целого числа
MCIERR_MISSING_PARAMETER	Команда ожидает требуемый параметр, который не указан
MCIERR_MULTIPLE	Возникли ошибки одновременно на нескольких устройствах
MCIERR_MUST_USE_SHAREABLE	Ошибка, связанная с монопольным доступом к драйверу MCI-устройства. Для установки совместного доступа следует использовать параметр `shareable`
MCIERR_NO_ELEMENT_ALLOWED	MCI-устройство не поддерживает работу с файлами
MCIERR_NO_WINDOW	Нет окна для вывода изображения. Укажите хендл окна в соответствующей команде
MCIERR_NONAPPLICABLE_FUNCTION	Нарушен порядок выполнения MCI-команд. Измените или пересмотрите порядок следования
MCIERR_NULL_PARAMETER_BLOCK	MCI-устройство требует в качестве параметра ненулевой аргумент
MCIERR_OUT_OF_MEMORY	Недостаточно памяти для выполнения MCI-команды
MCIERR_OUTOFRANGE	Указанный фрейм лежит вне допустимых границ области
MCIERR_SET_CD	MCI-устройство недоступно, т. к. приложение не может выбрать необходимый каталог
MCIERR_SET_DRIVE	Ошибка при смене драйвера необходимого MCI-устройства
MCIERR_UNNAMED_RESOURCE	Имя не указано

Таблица 3.9 *(окончание)*

Код ошибки	Описание и способ устранения
MCIERR_UNRECOGNIZED_COMMAND MCIERR_UNSUPPORTED_FUNCTION	Неизвестная команда. Проверьте, поддерживает ли данное MCI-устройство выбранные команды
MCIERR_SEQ_DIV_INCOMPATIBLE	Несовместимые форматы указателя "song pointer" и SMPTE
MCIERR_SEQ_NOMIDIPRESENT	Система не обнаружила устройства MIDI
MCIERR_SEQ_PORT_INUSE	Указанный порт MIDI уже используется
MCIERR_SEQ_PORT_MAPNODEVICE MCIERR_SEQ_PORT_NONEXISTENT	Устройство MIDI, которое не установлено или не найдено в системе
MCIERR_SEQ_PORT_MISCERROR	Ошибка порта MIDI
MCIERR_SEQ_PORTUNSPICIFIED	В системе не выбран порт MIDI. Установите порт в "Панель управления \ Звуки и аудиоустройства"
MCIERR_SEQ_TIMER	Все таймеры системы мультимедиа задействованы другими приложениями
MCIERR_WAVE_INPUTSINUSE	Задействованы все звуковые устройства, которые могут записывать файл в данном формате
MCIERR_WAVE_INPUTSUNSUITABLE	В системе не установлено ни одного устройства, которое могло бы записывать файл в данном формате
MCIERR_WAVE_INPUTUNSPECIFIED	Не указано устройство записи звука по умолчанию. Необходимо указать любое совместимое устройство записи звука в настройках "Панель управления \ Звуки и аудиоустройства"
MCIERR_WAVE_OUTPUTSINUSE	Задействованы все звуковые устройства, которые могут проигрывать файл в данном формате
MCIERR_WAVE_OUTPUTSUNSUITABLE	Не указано устройство воспроизведения звука по умолчанию. Необходимо указать любое совместимое устройство воспроизведения звука в настройках "Панель управления \ Звуки и аудиоустройства"
MCIERR_WAVE_SETINPUTINUSE	Текущее звуковое устройство уже задействовано
MCIERR_WAVE_SETINPUTUNSUITABLE MCIERR_WAVE_SETOUTPUTUNSUITABLE	Устройство, использованное для записи (воспроизведения) звука, не может распознать формат поступающих данных

Заключение

В данной главе мы коротко представили вашему вниманию краткое описание интерфейса Windows для работы с мультимедийными устройствами. В следующей главе мы с вами рассмотрим очень интересный, с точки зрения прикладного программиста, интерфейс Video for Windows — интерфейс для работы с видео.

ГЛАВА 4

Video for Windows – интерфейс для работы с видео

Краткий экскурс

Все мы поклонники кино. И пока критики спорят о достоинствах того или иного фильма, мы, зрители, с удовольствием смотрим его. С развитием цифровых технологий, а также повышением вычислительных мощностей просмотр фильмов на компьютере стал повседневным делом. Не важно, смотрим мы на компьютере видеоклипы или наслаждаемся новым блокбастером, — мы имеем дело с видеофайлами, представленными в Windows файлами с расширением avi, mpg, mov и др. В данной главе мы рассмотрим работу с AVI-файлами.

С концептуальной точки зрения, видеофайл представляет собой файл, хранящий последовательность видеокадров, задающих необходимый видеоряд. К последовательности видеокадров привязана аудиодорожка, синхронизированная с видеорядом по ключевым кадрам (фреймам). Видеоролик может содержать любое количество ключевых кадров, в зависимости от того, насколько часто и сильно меняется изображение. Эти кадры служат в своем роде опорными точками между переходами сцен. Ключевые кадры чрезвычайно важны при чтении видеоролика, без них промежуточные кадры теряют смысл, т. к. они, в свою очередь, представляют собой лишь изменение предыдущего изображения, а не самостоятельное (цельное) изображение.

Стоит также заметить, что наличие аудиодорожки в видеофайле необязательно. Более того, к видеоряду может быть привязано несколько аудиодорожек. Дополнительно видеофайл может содержать в себе различную текстовую информацию, а также музыкальный MIDI-поток. Такая универсальность достигается за счет определенной структуры видеофайла, позволяющей осуществлять раздельное хранение дорожек в так называемых потоках (streams). Поток представляет собой непрерывную последовательность цифровых данных, следующих друг за другом. Так, например, видеоряд представлен последовательностью кадров, следующих один за другим.

Важно также обратить внимание на то, что аудио- и видеопотоки могут быть представлены в видеофайле в разных форматах сжатия. В настоящее время существует множество различных алгоритмов сжатия, разработанных разными компаниями; каждый из них обладает своими достоинствами и недостатками. По этой причине разработчиками формата AVI и была предусмотрена встроенная поддержка различных форматов сжатия в потоках. Так, самым распространенным на сегодняшний день форматом сжатия видео является формат MPEG4 или DivX. Для аудио самым распространенным является MP3.

Такова вкратце структура файлов AVI, для работы с которыми предусмотрен интерфейс программирования Video for Windows.

Введение в Video for Windows

Video for Windows, в дальнейшем VFW — это интерфейс прикладного программирования (application programming interfaces), предоставляющий разработчикам набор функций и процедур для создания, редактирования и воспроизведения видеофайлов. Поддержка VFW изначально встроена в операционные системы Microsoft Windows.

VFW представляет собой ряд динамических библиотек, расположенных в системных каталогах Windows. Использование динамических библиотек предоставляет широкие возможности по вызову их функций из различных языков программирования, в том числе из Delphi, с помощью подключения соответствующих заголовочных файлов, так называемых оберток динамических библиотек.

Имена многих функций, представленных в VFW, начинаются с префикса `AVI`. Другие функции (например, относящиеся к сжатию) начинаются с префикса `IC`. Существуют некоторые исключения, но большинство функций VFW снабжены одним из этих префиксов. Так, например, функция открытия файла AVI имеет имя `AVIFileOpen`, а функция получения информации о файле AVI — `AVIFileInfo`.

Установка и требования к работе

К сожалению, в Delphi нет готовых заголовочных файлов VFW. Однако эта проблема уже решена различными энтузиастами, которые своими силами осуществляют перевод различных заголовочных файлов в Delphi. Так, например, заголовочный файл API VFW (обертку динамической библиотеки AviFile32.dll) вы можете взять с прилагаемого компакт-диска из каталога Install\VFW Headers.

Стоит сразу отметить, что в данном каталоге лежат две различных версии заголовочных файлов, это AVIFile32.pas и VFW.pas. Хотя эти версии по сути своей представляют собой одно и то же, в дальнейшем в наших примерах мы будем использовать версию обертки, представленную в файле AVIFile32.pas.

Для работы с VFW в Delphi вы должны подключить заголовочный файл AVIFile32 в секцию uses модуля вашей программы. Для использования некоторых функций и констант требуется наличие подключенного файла MMSystem (Win32 multimedia API Interface), который входит в поставку Delphi. Пример исходного кода вы можете увидеть в листинге 4.1.

Листинг 4.1. Подключение заголовочного файла

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, ComCtrls, ExtCtrls, ToolWin,
  AVIFile32,MMSystem;
```

Работа с VFW начинается с инициализации библиотеки с помощью функции AVIFileInit и заканчивается закрытием VFW с помощью AVIFileExit. Это основное правило, которое необходимо запомнить.

Практика использования

После краткого знакомства с VFW перейдем к практическим вопросам.

Открытие файлов AVI

Для работы с конкретным файлом AVI требуется для начала его открыть с помощью функции AVIFileOpen, имеющей параметры, представленные в табл. 4.1.

Таблица 4.1. *Параметры* *AVIFileOpen*

Параметр	Описание
ppfile	Указатель на структуру PAVIFILE, в которую будет помещен указатель на интерфейс IaviFile
szFile	Имя файла для открытия

Таблица 4.1 *(окончание)*

Параметр	Описание
Mode	Параметр определяет режим доступа, в котором файл будет открыт. Ниже представлены варианты доступа: OF_CREATE — создать файл. Если файл существует, то он будет обнулен; OF_DENYNONE — открывает файл без разделения доступа; OF_DENYREAD — открывает файл и запрещает его чтение другими процессами; OF_DENYWRITE — открывает файл и запрещает запись в него другими процессами; OF_EXCLUSIVE — открывает файл с монопольным доступом. Не один другой процесс не сможет открыть его, пока не будет выполнено закрытие файла; OF_READ — открывает файл только для чтения; OF_READWRITE — открывает файл для чтения и записи; OF_WRITE — открывает файл только для записи.
pclsidHandler	Указатель на класс стандартного или пользовательского заголовка. Если значение равно Nil — автоматически выбирается формат заголовка стандарта RIFF

Сразу стоит отметить, что по окончании работы с AVI-файлом его необходимо закрыть, для этого нужно воспользоваться функцией AVIFileRelease с параметром, указывающим на структуру PAVIFILE.

Пример правильного открытия файла вы можете увидеть в листинге 4.2.

Листинг 4.2. Правильное открытие и закрытие видеофайла

```
var
 hAvi : PAVIFile;
begin
 // Инициализация API
 AVIFileInit;
 // Открытие файла
 AVIFileOpen (hAvi, PChar(FilenameEdit.Text), OF_READ, nil);
 // * * * ЧТО-ТО ДЕЛАЕМ * * *
 // Все выгружаем и закрываем
 AVIFileRelease(hAvi); // Закрытие файла
 AVIFileExit; // Закрытие API
end;
```

Получение информации из заголовка файла AVI

Для получения информации о файле AVI из заголовка в VFW предназначена функция `AVIFileInfo`, имеющая следующий вид:

```
AVIFileInfo(pfile: IAVIFile; var pfi:
     TAVIFILEINFO; lSize: LONG) : LongInt
```

Данная функция имеет следующие параметры, представленные в табл. 4.2.

Таблица 4.2. *Параметры* `AVIFileInfo`

Параметр	Описание
`pfile`	Дескриптор AVI-файла
`Pfi`	Указатель на структуру `TAVIFileInfo`, куда будет занесена информация, взятая из заголовка AVI
`lSize`	Размер структуры `TAVIFileInfo`, который можно получить с помощью `sizeof(AVIFILEINFO)`

Структура `TAVIFileInfo` имеет следующие поля, представленные в табл. 4.3.

Таблица 4.3. *Поля структуры* `TAVIFileInfo`

Поле	Описание
`dwMaxBytesPerSec`	Максимальная скорость вывода потока видео, так называемый битрейт (bitrate), измеряется в битах в секунду (BPS)
`dwFlags`	Флаг, указывающий на дополнительную информацию о видеофайле, может принимать следующие значения: `AVIF_HASNDEX` — значение флага указывает на то, что данный видеофайл имеет индекс в конце файла; `AVIF_MUSTUSEINDEX` — значение флага указывает на то, что порядок воспроизведения задан в индексе, т. е. воспроизведение видео должно идти в соответствии с индексом, а не с физическим порядком следования видеоданных; `AVIF_ISINTERLEAVED` — значение флага указывает на то, что фильм записан в "чересстрочной" развертке (INTERLEAVED). Каждый кадр выводится не просто цельной картинкой, а картинкой, разбитой через каждую строку двумя полями. Такой способ вывода называется *интерлейсным* (дословно "чередование"). Данный вид развертки дает нам не 25, а 50 полукадров в секунду,

Таблица 4.3 *(продолжение)*

Поле	Описание
	что приводит к плавному выводу при смене изображения. Данный флаг используется для файлов, предназначенных для TV-вещания, т. к. монитор, в отличие от телевизора, выводит каждый кадр целиком, без чересстрочного вывода полей. Такой способ вывода называется *прогрессивным* (дословно "последовательным"); AVIF_WASCAPTUREFILE — флаг, указывающий на то, что данный видеофайл является захваченным (Capture) видео в режиме реального времени; AVIF_COPYRIGHTED — очень интересный флаг, указывающий, что AVI-файл содержит защищенные авторским правом данные. Теоретически программное обеспечение не должно разрешать копирование и дублирование видеоданных. Хотя за годы работы мы так и не встретили такого программного обеспечения
dwCaps	Флаг способностей (Дословно "Specifies capability flags"). В настоящее время данный флаг не используется
dwStreams	Определяет количество потоков в файле. Например, файл с аудио и видео имеет два потока
dwSuggestedBufferSize	Определяет рекомендуемый буферный размер для чтения и воспроизведения файла. Вообще, этот размер должен быть достаточно большим, чтобы содержать наибольший кусок видеоданных в файле. Если значение его равно 0 или является слишком маленьким, программное обеспечение воспроизведения видео должно будет само перераспределять оперативную память в течение времени воспроизведения. Для interleaved-файла этот буферный размер должен быть достаточно большим, чтобы читать кадр, идущий в двух строках развертки, а не только кусок его
dwWidth	Размер изображения видео по ширине в пикселах. Хочется сразу отметить, что в редких случаях размер, указанный в заголовке файла AVI, может отличаться от реального размера изображения в видеопотоке. Для правильного определения размера лучше всего воспользоваться функцией получения информации о потоке, о которой будет рассказано далее
dwHeight	Размер видеоизображения по высоте в пикселах
dwScale	Значение, представленное в этом поле, используется с dwRate, чтобы определить скорость воспроизведения видео и аудио в масштабе времени. Стоит сразу заметить, что в соответствии с архитектурой AVI каждый поток может иметь собственный масштаб времени воспроизведения.

Таблица 4.3 *(окончание)*

Поле	Описание
	Деление значения `dwRate` на `dwScale` дает скорость воспроизведения. Для видео это значение измеряется в фреймах (кадрах) в секунду (FramesPerSec), для аудио — в блоках в секунду (BlocksPerSec)
`dwRate`	Значение задает темп (блок/секунда)
`dwLength`	Определяет длину времени воспроизведения AVI-файла. Единицы определены в `dwRate` и `dwScale`
`dwEditCount`	Определяет количество потоков, которые были добавлены или удалены
`szFileType`	В данном поле содержится описательная часть информации о видеофайле

Для демонстрации использования функции `AVIFileInfo` мы разработали программу, часть исходного кода которой представлена в листинге 4.3. Полную версию программы вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch04\Ex01.

Листинг 4.3. Получение информации о видеофайле из заголовка AVI

```
procedure TForm1.FileListBoxChange(Sender: TObject);
var
 hAvi    : PAVIFile;
 infoAvi : TAVIFileInfo;
begin
 IF FileListBox.FileName = '' Then Exit;
 Try
  // Инициализация и открытие файла
  AVIFileInit; // Инициализация API
  AVIFileOpen (hAvi, PChar(FileListBox.FileName), OF_READ, nil);
  // Информация, взятая из заголовка
  AVIFileInfo (hAvi, infoAvi, SizeOf(infoAvi)); // Получение информации
  with InfoAvi do
  begin
      // Размер
      Size.Caption       := IntToStr(dwWidth) + ' x ' +
                            IntToStr(dwHeight);
      // FPS
      FPSHeader.Caption  := IntToStr(Round(dwRate/dwScale));
```

```
      // Время
      TimeHeader.Caption := TimeToStr(dwLength/(dwRate/dwScale) /
                                      SecsPerDay);
  end;
  // Все выгружаем и закрываем
  AVIFileRelease (hAvi); // Закрытие файла
  AVIFileExit; // Закрытие API
 except
 end;
end;
```

Обратите внимание, что время воспроизведения видеофайла представлено в привычных человеку единицах измерения, а именно в ЧЧ.ММ.СС.

Конечно, на этом этапе мы получили лишь малую часть информации о видеофайле. Для полноты картины хотелось бы также узнать, с помощью какого кодека были зажаты видео- и аудиоданные в файле, а также степень сжатия аудио. Получить эту информацию из заголовка AVI-файла с помощью `AVIFileInfo` нельзя, зато, используя функции чтения информации о потоках, узнать эти данные не составит особых проблем. Однако перед считыванием этих данных мы должны обеспечить доступ к потокам.

Доступ к потокам

Для получения доступа к потокам, находящимся в видеофайле, нужно открыть поток с помощью функции `AVIFileGetStream`:

```
function AVIFileGetStream(pFile: PAVIFile;
             var pStream: PAVIStream;
             fccType:
             FourCC; lParam: LongInt) : LongInt;
```

Параметры функции представлены в табл. 4.4.

После работы с потоком его нужно обязательно закрыть. Закрытие потока происходит с помощью функции `AVIStreamRelease`.

Таблица 4.4. *Параметры* `AVIFileGetStream`

Параметр	Описание
`pfile`	Дескриптор AVI-файла
`ppavi`	Дескриптор потока

Таблица 4.4 *(окончание)*

Параметр	Описание
fccType	Описатель вида потока. Данный параметр может принимать следующие значения: StreamtypeAUDIO — аудиопоток; StreamtypeVIDEO — видеопоток; StreamtypeMIDI — музыкальный MIDI-поток; StreamtypeTEXT — текст; а также значение Nil, определяющее перечисление всех потоков
lParam	Порядковый номер потока при перечислении

Следующий пример, представленный в листинге 4.4, демонстрирует открытие видеопотока.

Листинг 4.4. Получение доступа к видеопотоку

```
  // Берем первый видеопоток
IF AVIFileGetStream (hAvi, hStreamVideo, STREAMTYPEVIDEO, 0) =
                     AVIERR_OK
Then begin
  // Производим какое-либо действие
end;
```

Получение информации о потоке

Для корректной работы с потоками программисту требуется получить информацию о потоке с помощью функции AVIStreamInfo:

```
function  AVIStreamInfo(pStream: PAVIStream;
           var asiInfo: TAVIStreamInfo;
           lSize: LongInt): LongInt
```

Данная функция имеет следующие параметры, представленные в табл. 4.5.

Таблица 4.5. *Параметры* AVIStreamInfo

Параметр	Описание
pavi	Дескриптор потока
AsiInfo	Указатель на структуру TAVIStreamInfo. Данная структура на выходе заполняется информацией о потоке

Таблица 4.5 *(окончание)*

Параметр	Описание
lSize	Размер структуры AVIStreamInfo, получить который можно с помощью sizeof(AVIStreamInfo)

Отметим, что структура AVIStreamInfo является базовой для каждого потока. В зависимости от типа потока, указанного в дескрипторе, структура будет иметь следующие значения полей, представленные в табл. 4.6. Так как нас в основном интересуют потоки видео и аудио, информация о потоках MIDI и текста рассматриваться нами не будет.

Таблица 4.6. *Поля структуры* AVIStreamInfo

Поле	Описание для видеопотока	Описание для аудиопотока
fccType	Тип потока	То же
fccHandler	32-битный код (FOURCC), идентифицирующий формат сжатия видеокодеком (компрессором сжатия видео). Подробнее о FOURCC будет рассказано далее	Не используется
dwFlags	Флаг потока может принимать следующие значения: AVISF_DISABLED — флаг указывает на то, что данный поток нельзя использовать по умолчанию; AVISF_VIDEO_PALCHANGES — флаг указывает на то, что видеопоток имеет измененную цветовую палитру. Наличие данного флага указывает программному обеспечению на необходимость смены палитры	Флаг потока. Назначение то же, за исключением AVISF_VIDEO_PALCHANGES
dwCaps	Флаги свойств. Не используются	То же
wPriority	Приоритет потока	То же
wLanguage	Не используется	Идентификатор языка потока. Используется для файлов, содержащих аудиодорожки на разных языках.

Таблица 4.6 *(продолжение)*

Поле	Описание для видеопотока	Описание для аудиопотока
		К примеру, если к видео прикреплены две аудиодорожки на разных языках, то с помощью `wLanguage` можно идентифицировать выбор дорожки для воспроизведения
`dwScale`	Масштаб времени потока. Разделив `dwRate` на `dwScale`, получим скорость потока в фреймах в секунду	Масштаб времени потока. Разделив `dwRate` на `dwScale`, для несжатых файлов получим скорость потока в блоках в секунду. При наличии сжатия `dwScale` будет равно полю `nBlockAlign` структуры `WAVEFORMATEX`
`dwRate`	Скорость в целочисленном значении. Для получения скорости кадров фреймов необходимо значение разделить на `dwScale`	Скорость в целочисленном значении. Для получения размера блока аудиоданных значение необходимо разделить на `dwScale`
`dwStart`	Номер фрейма, с которого начинается воспроизведение (это важно, потому что первый кадр видеоролика может иметь индекс 0 или 1)	То же
`dwLength`	Определяет длину времени воспроизведения потока. Единицы определены в `dwRate` и `dwScale`	То же
`dwInitialFrames`	Не используется	Сдвиг аудиодорожки. Этот параметр определяет, через какое время надо начать воспроизведение видео после начала аудио. Значение обычно равно 3/4 (0,75) секунды
`dwSuggested BufferSize`	Желательный размер буфера памяти в байтах. Обычно содержит размер наибольшей цепочки потока. Равен нулю, если неизвестен желательный размер	То же

Таблица 4.6 *(окончание)*

Поле	Описание для видеопотока	Описание для аудиопотока
`dwQuality`	Качество записи от 0 до 10 000. Для сжатых данных значение равно параметру качества, переданному кодеку сжатия. Если было передано сжатие по умолчание, то значение равно –1	То же
`dwSampleSize`	Размер одного фрейма данных в байтах. Если параметр равен нулю, то размер может изменяться. Например, в сжатых видеоданных размеры фрейма в зависимости от картинки могут варьироваться	Размер одного блока аудио. Значение равно полю `nBlockAlign` структуры `WAVEFORMATEX`
`rectFrame`	Размеры окна видеоизображения. `RectFrame` хранит координаты верхнего-левого угла + высоту и ширину	Не используется
`dwEditCount`	Количество редакций потока	То же
`dwFormatChange Count`	Количество изменений формата потока	То же
`szName`	Описание потока	То же

Как видим, данная структура предоставляет только общую информацию о потоке и не несет информации о самих данных. Однако перед тем как мы перейдем к получению и разбору функции, предоставляющей информацию о потоке на основе анализа данных, временно остановимся на поле `fccHandler`.

Поле `fccHandler` используется для получения информации о способе сжатия видеоданных и применяется при выборе соответствующего кодека. Идентификация кодека основывается на применении так называемой системы обозначения кодека посредством четырехбуквенного кода (FourCC — Four Character Code). Коды различных кодеков жестко заданы компанией Microsoft. Например, FOURCC 'CVID' идентифицирует кодек Cinepak (прежде Compact Video).

Примечание

Стоит также отметить, что Microsoft использует FOURCC также для уникальной идентификации представления пиксела, используемого в несжатых изображениях и видео. Например, коды 'YUY2' идентифицируют представление пикселов в YUV-модели представления цвета. Возьмем данное примечание на заметку.

Как было сказано, любой кодек сжатия видео должен быть зарегистрирован в Microsoft, однако практика показывает, что по тем или иным причинам многие кодеки не имеют такой регистрации. В их числе популярный на сегодняшний день DivX.

В табл. 4.7 представлен список идентификаторов зарегистрированных кодеков. Возможно, на момент публикации данной книги этот список окажется уже не полным.

Таблица 4.7. *FOURCC коды кодеков*

Код	Кодек	Код	Кодек
3iv1	3ivx Delta 1/2/3	H265	Intel ITU H.265
3iv2	3ivx Delta 4	H266	Intel ITU H.266
aasc	Autodesk Animator	H267	Intel ITU H.267
afli	Autodesk Animator	H268	Intel ITU H.268
aflc	Autodesk Animator	H269	Intel ITU H.269
ap41	AngelPotion	i263	Intel ITU H.263
asv1	Asus Video	ir21	Intel Indeo 2.1
asv2	Asus Video 2	iv30	Ligos Indeo 3
bink	Bink Video	iv31	Ligos Indeo 3
cram	Microsoft Video 1	iv32	Ligos Indeo 3.2
cvid	Cinepak Radius	iv33	Ligos Indeo 3
div3	DivX 3.11 Low Motion	iv34	Ligos Indeo 3
div4	DivX 3.11 Fast Motion	iv35	Ligos Indeo 3
div5	DivX 3.11 VKI Low Motion	iv36	Ligos Indeo 3
div6	DivX 3.11 VKI Fast Motion	iv37	Ligos Indeo 3
divx	DivX 4	iv38	Ligos Indeo 3
dmb1	Matrox Rainbow Runner	iv39	Ligos Indeo 3
dx50	DivX 5	iv40	Ligos Indeo Inderactive 4
H260	Intel ITU H.260	iv41	Ligos Indeo Inderactive 4
H261	Intel ITU H.261	iv42	Ligos Indeo Inderactive 4
H262	Intel ITU H.262	iv43	Ligos Indeo Inderactive 4
H263	Intel ITU H.263	iv44	Ligos Indeo Inderactive 4
H264	Intel ITU H.264	iv45	Ligos Indeo Inderactive 4

Таблица 4.7 *(окончание)*

Код	Кодек	Код	Кодек
iv46	Ligos Indeo Inderactive 4	MP4S	Microsoft MPEG-4
iv47	Ligos Indeo Inderactive 4	MPEG	MPEG-1
iv48	Ligos Indeo Inderactive 4	MPG4	Microsoft MPEG-4 High Speed Compressor
iv49	Ligos Indeo Inderactive 4	MRLE	Microsoft RLE
lv50	Ligos Indeo Inderactive 5	MSVC	Microsoft Video 1
m261	Microsoft H.261	rv20	RealVideo G2
m263	Microsoft H.263	rv30	RealVideo 8
MC12	ATI Motion Compensation Format	tscc	TechSmith Screen Capture
MCAM	ATI Motion Compensation Format	VCR1	ATI Video Codec 1
MJPG	Motion JPEG	VCR2	ATI Video Codec 2
MP42	Microsoft MPEG-4	XMPG	Xing MPEG
MP43	Microsoft MPEG-4	XVID	XviD

Часто в различных конференциях и форумах в Интернете, посвященных работе с видео, начинающие программисты задают один и тот же вопрос: "Каким способом можно получить буквенное обозначение FourCC на основе числового значения поля `fccHandler`?". Для ответа на этот вопрос в листинге 4.5 мы представили нашим читателям функцию `FourCCToString`, предназначенную для преобразования числового кода в буквенный.

Листинг 4.5. Преобразование кода FourCC из численного в буквенный

```
Function FourCCToString(lFourCC : DWORD): String;
var
   sRet   : String;
   lUByte : DWORD;
begin
   sRet := Chr(lFourCC And $FF);
   sRet := sRet + Chr((lFourCC And $FF00) div $100);
   sRet := sRet + Chr((lFourCC And $FF0000) div $10000);
   lUByte := (lFourCC And $7F000000) div $1000000;
   If (lFourCC And $80000000) = $80000000 Then
       lUByte := lUByte Or $80;
```

```
   sRet := sRet + Chr(lUByte);
   FourCCToString := sRet;
End;
```

Продолжим. Для получения информации о потоке на основе анализа формата данных в VFW предусмотрена функция `AVIStreamReadFormat`:

```
function AVIStreamReadFormat(pStream: PAVIStream; lPos: LongInt;
              lpFormat: LPVoid;
              var cbFormat: LongInt):LongInt
```

Параметры функции представлены в табл. 4.8.

Таблица 4.8. *Параметры* `AVIStreamReadFormat`

Параметр	Описание
`pavi`	Дескриптор потока
`lPos`	Позиция в потоке, для которой считывается формат. Обычно достаточно считать формат только для первой позиции
`lpFormat`	Указатель на область памяти, которая будет заполнена информацией о потоке
`lpcbFormat`	Размер блока памяти для заполнения. Если параметр `lpFormat` будет равен `Nil`, то после вызова параметр `lpcbFormat` примет значение необходимого размера области памяти. То есть, для получения необходимого размера следует вызвать следующий код: `AVIStreamReadFormat (hStream, 0, nil, hdrSize);`

После получения необходимого размера области памяти мы можем получить информацию о потоке на основе анализа формата данных, однако перед этим мы должны уяснить следующее. Дело в том, что в зависимости от типа потока функция `AVIStreamReadFormat` должна заполнять разные структуры. Так, для видеопотока в качестве параметра `lpFormat` должен быть передан указатель на структуру `BITMAPINFOHEADER`, а для аудиопотока — на структуру `WAVEFORMATEX`. Также перед вызовом функции `AVIStreamReadFormat` для этих структур должна быть выделена область памяти с помощью функции `GetMem`.

В листинге 4.6 показан пример получения подробной информации об аудиопотоке.

Листинг 4.6. Получение служебной информации об аудиопотоке

```
var
 hAvi, hStream : PAVIFile;
```

```
 hdrSize       : Integer;
 hdrWav        : PWaveFormat; // Указатель на WAVEFORMATEX
begin
 // Инициализация и открытие файла
 AVIFileInit; // иниц. API
 AVIFileOpen (hAvi, PChar(FilenameEdit.Text), OF_READ, nil);

 // Берем первый аудиопоток
 AVIFileGetStream (hAvi, hStream, streamtypeAUDIO, 0);

 // Подробная информация о звуковом потоке на основе анализа формата
 AVIStreamReadFormat (hStream, 0, nil, hdrSize);          // Получение
длины
                                                          // блока памяти
 GetMem (hdrWav, hdrSize);                                // Выделение памяти
 // Заполнение структуры WAVEFORMATEX
 AVIStreamReadFormat (hStream, 0, hdrWav, hdrSize);
 // ***** ЧТО ТО ДЕЛАЕМ ******
 AVIStreamRelease (hStream);                          // Закрываем поток
 FreeMem (hdrWav);                                    // Освобождаем память
 // Все выгружаем и закрываем
 AVIFileRelease (hAvi); // Закрытие файла
 AVIFileExit; // Закрытие API
end;
```

Как видите, в данном коде нет ничего сложного, однако использование функции `AVIStreamReadFormat` следует рассматривать в комплексе с разбором структур `BITMAPINFOHEADER` и `WAVEFORMATEX`. Структура `BITMAPINFOHEADER` определена в Win32 API и имеет следующие поля, представленные в табл. 4.9. Некоторые поля структуры (например, `biXPelsPerMeter` и `biYPelsPerMeter`) не имеют отношения к AVI-файлам, но таких полей немного, поэтому в таблице будет дано полное описание структуры `BITMAPINFOHEADER`.

Таблица 4.9. *Поля структуры* `BITMAPINFOHEADER`

Поле	Описание
`biSize`	Число байтов, требующихся в соответствии со структурой
`biWidth`	Ширина растрового изображения в пикселах
`biHeight`	Высота растрового изображения в пикселах
`biPlanes`	Число плоскостей для целевого устройства. Обычно оно равно 1

Таблица 4.9 *(окончание)*

Поле	Описание
biBitCount	Число битов на пиксел. Поле может принимать следующие значения: 8, 16, 24 или 32
biCompression	Тип уплотнения. Поле может принимать следующие значения: BI_RGB — несжатое; BI_RLE8 — формат кодирования длин серий (RLE — run-length encoding) для растра изображений с 8 битами на пиксел. Формат сжатия представляет собой двухбайтный формат, состоящий из байта числа повторений, за которым следует байт, содержащий индекс цвета; BI_RLE4 — формат RLE для растровых изображений с 4 битами на пиксел. Формат сжатия представляет собой двухбайтный формат, состоящий из байта числа повторений, за которым следует байт, содержащий индекс цвета; BI_BITFIELDS — указывает, что растровое изображение не сжато и таблица цветов состоит из трех масок цвета в виде двойного слова, которые содержат, соответственно, красный, зеленый и синий компонент каждого пиксела, Данное значение параметра является допустимым при его использовании для растровых изображений с 16 и 32 битами на пиксел; BI_JPEG — указывает, что изображение имеет формат JPEG; BI_PNG — указывает, что изображение имеет формат PNG
biSizeImage	Размер изображения в байтах
biXPelsPerMeter	Разрешающая способность по горизонтали целевого устройства растрового изображения в пикселах на метр
biYPelsPerMeter	Разрешающая способность по вертикали целевого устройства растрового изображения в пикселах на метр
biClrUsed	Число индексов цвета в таблице цветов, фактически используемых в растровом изображении. Если это значение равно 0, в растровом изображении используется максимальное число цветов, соответствующее значению поля biBitCount для режима сжатия, указанного параметром biCompression
biClrImportant	Число индексов цвета, которые необходимы для вывода растрового изображения. Если это значение равно 0, необходимы все цвета

Структура WAVEFORMATEX определяет информацию о формате аудиоданных. Данная структура включена в Win32 Multimedia Programmer's Reference API и имеет следующие поля, указанные в табл. 4.10.

Таблица 4.10. *Поля структуры* `WAVEFORMATEX`

Поле	Описание
`wFormatTag`	Идентифицирующий код, указывающий на формат сжатия аудио. Так же, как и для видео, для аудио корпорацией Microsoft определены индивидуальные коды для многих алгоритмов сжатия. Для несжатых данных используется `WAVE_FORMAT_PCM`. Подробный список идентификаторов вы можете найти в табл. 4.11
`nChannels`	Количество каналов в аудиоданных. Монофонические данные используют один канал, стереоданные — два канала
`nSamplesPerSec`	Скорость выборок в секунду — так называемая частота дискретизации. Если `wFormatTag` имеет значение `WAVE_FORMAT_PCM` (далее PCM), то обычно поле `nSamplesPerSec` принимает следующие значения: 8,0 кГц, 11,025 кГц, 22,05 кГц, и 44,1 кГц. Если аудиоданные сжаты каким-либо аудиокодеком, то значение `nSamplesPerSec` должно быть вычислено согласно спецификации изготовителя данного кодека
`nAvgBytesPerSec`	Расчетная скорость передачи данных в байтах. Если формат PCM, то `nAvgBytesPerSec` должен быть равен `nSamplesPerSec` × `nChannels`. Для форматов не PCM значение должно быть вычислено согласно спецификации изготовителя
`nBlockAlign`	Выравнивание блока данных. Очень важная переменная для работы со сжатыми данными. Дело в том, что, в отличие от PCM-данных, многие кодеки работают с относительно большими блоками, разделение которых на части невозможно. Эта переменная описывает размер блока, необходимого кодеку для работы. Для PCM-данных она равна `nChannels`×`wBitsPerSample/8`, что равно размеру одной выборки. Для форматов не PCM значение должно быть вычислено согласно спецификации изготовителя
`wBitsPerSample`	Количество бит на одну выборку. Для PCM `wBitsPerSample` может принимать значения 8 или 16. Для форматов не PCM значение поля должно быть установлено согласно спецификации изготовителя. Также следует учесть, что некоторые форматы сжатия не могут определить значение `wBitsPerSample`, так что в некоторых случаях значение может быть нулевым
`cbSize`	Размер блока дополнительных данных, добавленных в конец структуры `WAVEFORMATEX`. Используется при сжатии и распаковке данных. Данное поле может использоваться некоторыми форматами, чтобы хранить дополнительные данные. Обратите внимание, что для форматов `WAVE_FORMAT_PCM` значение данного поля игнорируется.

Таблица 4.10 *(окончание)*

Поле	Описание
	Пример использования данного поля хорошо виден в аудиокодеке MS-ADPCM (идентификатор `WAVE_FORMAT_ADPCM`). В данном кодеке `cbSize` всегда будет равен 32. Дополнительная информация для этого кодека будет содержать коэффициенты, требующиеся для зашифровки и расшифровки звуковых данных

В табл. 4.11 представлен список идентификаторов (в шестнадцатеричной системе счисления) зарегистрированных аудиокодеков, значения которых могут быть представлены в поле `wFormatTag`.

Таблица 4.11. *Коды аудиокодеков*

Код	Кодек	Код	Кодек
0001	PCM	0020	YAMAHA ADPCM
0002	ADPCM	0021	SONARC
0003	IEEE FLOAT	0022	DSPGROUP TRUESPEECH
0004	VSELP	0023	ECHOSC1
0005	IBM CVSD	0024	AUDIOFILE AF36
0006	ALAW	0025	APTX
0007	MULAW	0026	AUDIOFILE AF10
0008	DTS	0027	PROSODY 1612
0009	DRM	0028	LRC
0010	OKI ADPCM	0030	DOLBY AC2
0011	DVI ADPCM / IMA ADPCM	0031	GSM610
0012	MEDIASPACE ADPCM	0032	MSNAUDIO
0013	SIERRA ADPCM	0033	ANTEX ADPCME
0014	G723 ADPCM	0034	CONTROL RES VQLPC
0015	DIGISTD	0035	DIGIREAL
0016	DIGIFIX	0036	DIGIADPCM
0017	DIALOGIC OKI ADPCM	0037	CONTROL RES CR10
0018	MEDIAVISION ADPCM	0038	NMS VBXADPCM
0019	CU CODEC	0039	CS IMAADPCM

Таблица 4.11 *(продолжение)*

Код	Кодек	Код	Кодек
003a	ECHOSC3	0079	VOXWARE TQ40
003b	ROCKWELL ADPCM	0080	SOFTSOUND
003c	ROCKWELL DIGITALK	0081	VOXWARE TQ60
003d	XEBEC	0082	MSRT24
0040	G721 ADPCM	0083	G729A
0041	G728 CELP	0084	MVI MVI2
0042	MSG723	0085	DF G726
0050	MPEG	0086	DF GSM610
0052	RT24	0088	ISIAUDIO
0053	PAC	0089	ONLIVE
0055	MPEG Layer 3 (MP3)	0091	SBC24
0059	LUCENT G723	0092	DOLBY AC3 SPDIF
0060	CIRRUS	0093	MEDIASONIC G723
0061	ESPCM	0094	PROSODY 8KBPS
0062	VOXWARE	0097	ZYXEL ADPCM
0063	CANOPUS ATRAC	0098	PHILIPS LPCBB
0064	G726 ADPCM	0099	PACKED
0065	G722 ADPCM	00a0	MALDEN PHONYTALK
0067	DSAT DISPLAY	0100	RHETOREX ADPCM
0069	VOXWARE BYTE ALIGNED	0101	IRAT
0070	VOXWARE AC8	0111	VIVO G723
0071	VOXWARE AC10	0112	VIVO SIREN
0072	VOXWARE AC16	0123	DIGITAL G723
0073	VOXWARE AC20	0125	SANYO LD ADPCM
0074	VOXWARE RT24	0130	SIPROLAB ACEPLNET
0075	VOXWARE RT29	0131	SIPROLAB ACELP4800
0076	VOXWARE RT29HW	0132	SIPROLAB ACELP8V3
0077	VOXWARE VR12	0133	SIPROLAB G729
0078	VOXWARE VR18	0134	SIPROLAB G729A

Таблица 4.11 *(окончание)*

Код	Кодек	Код	Кодек
0135	SIPROLAB KELVIN	0251	IPI RPELP
0140	G726ADPCM	0260	CS2
0150	QUALCOMM PUREVOICE	0270	SONY SCX
0151	QUALCOMM HALFRATE	0300	FM TOWNS SND
0155	TUBGSM	0400	BTV DIGITAL
0160	MSAUDIO1	0450	QDESIGN MUSIC
0161	DivX Audio (WMA 2)	0680	VME VMPCM
0170	UNISYS NAP ADPCM	0681	TPC
0171	UNISYS NAP ULAW	1000	OLIGSM
0172	UNISYS NAP ALAW	1001	OLIADPCM
0173	UNISYS NAP 16K	1002	OLICELP
0200	CREATIVE ADPCM	1003	OLISBC
0202	CREATIVE FASTSPEECH8	1004	OLIOPR
0203	CREATIVE FASTSPEECH10	1100	LH CODEC
0210	UHER ADPCM	1400	NORRIS
0220	QUARTERDECK	1500	SOUNDSPACE MUSICOMPRESS
0230	ILINK VC	2000	AC3
0240	RAW SPORT	fffe	EXTENSIBLE (РАСШИРЯЕМЫЙ)
0241	ESST AC3	ffff	DEVELOPMENT (РАЗВИВАЮЩИЙСЯ)
0250	IPI HSX		

Помимо основных функций `AVIStreamInfo` и `AVIStreamReadFormat`, в VFW есть еще несколько простых функций:

- `AVIStreamLength` — позволяет получить длину потока. Для видеопотока длина возвращается в кадрах. Для звуковых потоков функция будет возвращать длину в блоках аудиоданных. Например, если звуковые данные имеют частоту 22 кГц и ADPCM — формат сжатия, то блок аудиоданных будет равен 256 байтам.

Примечание

Для получения длины потока в миллисекундах можно использовать функцию `AVIStreamSampleToTime`.

- `AVIStreamStart` — позволяет получить начальную позицию потока. Так как начальная позиция потока в AVI-файле жестко не регламентирована, следует использовать данную функцию для получения реальной стартовой позиции.

Как видите, в получении информации о потоках AVI-файла нет ничего сложного. Для закрепления материала мы разработали демонстрационную программу, которая находится на прилагаемом компакт-диске в каталоге Source\Ch04\Ex01. На рис. 4.1 представлен результат работы данной программы.

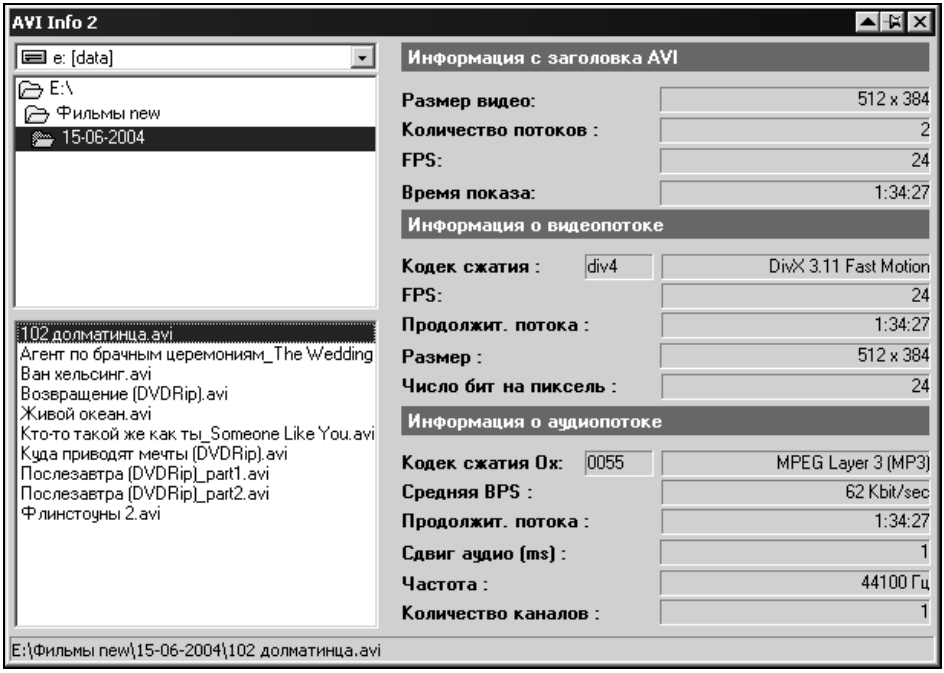


Рис. 4.1. Программа Avi Info 2

Работа с кадрами. Сохранение отдельных кадров в формат BMP

Следующим шагом в осваивании Video For Windows станет изучение работы с отдельными кадрами видеоизображения. Технология работы с кадрами в VFW достаточно проста. Для того чтобы считать некоторый кадр из файла AVI, программистам на VFW требуется сначала выполнить декомпрессию (распаковку) кадра, т. е. подготовить кадр к отображению. Делается это с по-

мощью следующих функций — AVIStreamGetFrameOpen, AVIStreamGetFrame и AVIStreamGetClose.

Функция AVIStreamGetFrameOpen запрашивает декомпрессор для выбранного видеопотока и, если таковой имеется, открывает его. Данная функция имеет следующий вид:

```
function AVIStreamGetFrameOpen(pStream: PAVIStream;
          pbmpWanted: PBitmapInfoHeader): PgetFrame
```

На входе функция принимает следующие параметры, представленные в табл. 4.12.

Таблица 4.12. *Параметры* `AVIStreamGetFrameOpen`

Параметр	Описание
pStream	Дескриптор потока
pbmpWanted	Указатель на структуру BITMAPINFOHEADER, определяющую желательный видеоформат вывода. Если значение равно Nil, формат будет установлен по умолчанию. Если же указать значение, равное AVIGETFRAMEF_BESTDISPLAYFMT, то будет выбран наилучший формат, что существенно повысит производительность. Это проще, чем инициализация структуры BITMAPINFOHEADER, но это означает, что вам неизвестно, какой формат будет получен на выходе — он определится текущими настройками экрана и аппаратной конфигурации компьютера

После выполнения функция AVIStreamGetFrameOpen возвращает объект GetFrame, либо Nil, если соответствующий декомпрессор видео не был найден.

После успешного считывания объекта GetFrame мы можем получить любой кадр потока в выбранном формате. Для этого нам нужно вызвать функцию AVIStreamGetFrame, имеющую следующий вид:

```
AVIStreamGetFrame(pgFrame: PgetFrame; lPos: LongInt): LPVoid
```

В качестве параметров функция принимает следующие значения, представленные в табл. 4.13.

Таблица 4.13. *Параметры* `AVIStreamGetFrame`

Параметр	Описание
pgFrame	Указатель на объект GetFrame, полученный при вызове AVIStreamGetFrameOpen
lPos	Номер фрейма (кадра)

После завершения работы функции на выходе в параметре LPVoid мы получим выбранный кадр в необходимом нам формате, либо Nil, если такого кадра не существует.

> ***Примечание***
>
> Заметим, что в самом начале полученного кадра идет структура BITMAPINFOHEADER, а уже следом сами данные.

Важной особенностью работы с функцией AVIStreamGetFrame является то, что полученный кадр будет действителен только до повторного вызова функции или до закрытия кадра с помощью функции AVIStreamGetFrameClose.

После работы с кадрами нужно не забыть освободить занятые ресурсы объекта GetFrame с помощью функции AVIStreamGetFrameClose.

Для демонстрации работы данных функций представляем демонстрационный проект, позволяющий просматривать кадры видео, а также сохранять выбранные кадры в графический файл формата BMP. При разработке данного проекта нами была поставлена задача не только показать читателю принципы работы с данными функциями, а также разобрать вопросы, связанные с сохранением отдельных кадров потока в растровый файл. Такой практический подход был обусловлен множеством вопросов, связанных с сохранением кадров, задаваемых в различных интернет-форумах и конференциях. В данном примере мы постарались дать на них ответ.

В листинге 4.7 сокращенно представлена функция GrabFrameAVI, позволяющая отобразить выбранный кадр, заданный в параметре iFrameNumber в объекте Image. Полный исходный код примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch04\Ex03.

Листинг 4.7. Отображение кадра в объекте Image

```
Procedure TForm1.GrabFrameAVI(AVIStream : PAVIStream;
                              iFrameNumber: Integer);
var
  gapgf  : PGETFRAME;
  lpbi   : PBITMAPINFOHEADER;
  bits   : PChar;
  hBmp   : HBITMAP;
  TmpBmp : TBitmap;
  DC_Handle : HDC;
begin
  // Подготовка к декомпрессии видеокадра
  gapgf := AVIStreamGetFrameOpen(AVIStream, nil);
```

```
  // Читаем фрейм
  // Функция AVIStreamGetFrame возвращает адрес
  // декомпрессированного видеокадра
  lpbi := AVIStreamGetFrame(gapgf, iFrameNumber);
  // Создаем Bitmap
  TmpBmp := TBitmap.Create;
  TmpBmp.Height := lpbi.biHeight;
  TmpBmp.Width  := lpbi.biWidth;
  // Массив байтов, содержащих начальные данные растрового изображения
  bits := Pointer(Integer(lpbi) + SizeOf(TBITMAPINFOHEADER));

  // Создаем контекст устройства дисплея
  DC_Handle := CreateDC('Display', nil, nil, nil);
  try
    // Создаем аппаратно-независимое растровое изображение (DIB)
    hBmp := CreateDIBitmap(DC_Handle, // Контекст устройства
                           lpbi^,     // Указатель на КАДР
                           CBM_INIT,  // Флаг, указывающий, что для
                                      // инициализации
                                      // растрового изображения
                                      // применяются данные
                                      // lpbi и  bits.
                           bits,      // Указатель на массив байтов,
                                      // содержащих
                                      // начальные данные растрового
                                      // изображения
                           PBITMAPINFO(lpbi)^, // Указатель на структуру
                                      // BitMapInfo, которая описывает
                                      // размеры и формат представления
                                      // цвета в массиве bits
                           DIB_RGB_COLORS); // Устанавливаем явные цвета
  finally
    DeleteDC(DC_Handle);
  end;
  TmpBmp.Handle := hBmp;

  // Закрываем фрейм
  AVIStreamGetFrameClose(gapgf);
  // Показываем изображение
  Image.Picture.Assign(TmpBmp);
  // Очищаем память
  TmpBmp.Free;
end;
```

Рис. 4.2. Работа с кадрами (сохранение отдельных кадров в формат BMP)

Результат работы демонстрационной программы вы можете увидеть на рис. 4.2.

Работа с кадрами. Сохранение BMP-файлов в AVI-формат

Одной из часто встречающихся задач при программировании VFW является задача создания AVI-файла по кадрам. Задача формулируется примерно так: есть кадры (заданные в формате BMP или генерируемые на экране), их, в свою очередь, нужно записать в файл AVI. Очень часто данная операция применяется в различных системах проектирования и анимации, а также в программах дизайна. В этом разделе мы разберем способ создания AVI-файла из разных изображений, имеющих одинаковую ширину, высоту и глубину цвета. Такая постановка задачи позволит упростить материал и сделать его более доступным для понимания. Однако при небольшом желании приведенный пример легко можно будет подогнать под свои нужды.

Итак, порядок действий примерно следующий: сначала нужно создать новый (пустой) файл AVI. Как мы уже разбирали, сделать это несложно, используя функцию `AVIFileOpen` с флагами `OF_CREATE` и `OF_WRITE`:

```
AVIFileOpen(tekFile,
     PChar(ExtractFilePath(Paramstr(0))+'\Demo.AVI'),
     OF_CREATE or OF_WRITE, nil);
```

Следующим шагом в работе после создания файла AVI — создание потока с помощью функции AVIFileCreateStream. Данная функция создает новый поток в существующем файле и возвращает указатель интерфейса потока pStream. Функция имеет следующий вид:

```
function AVIFileCreateStream(pFile: PAVIFile;
             var pStream: PAVIStream;
             var asiInfo: TAVIStreamInfo): LongInt
```

Параметры функции представлены в табл. 4.14.

Таблица 4.14. *Параметры* `AVIFileCreateStream`

Параметр	Описание
pFile	Дескриптор AVI-файла
pStream	Дескриптор потока
asiInfo	Указатель на структуру TAVIStreamInfo, содержащую информацию о новом потоке. Данная структура перед вызовом функции должна быть инициализирована

Теперь, когда переменная pStream указывает на поток, мы можем использовать ее для создания сжатого потока, содержащего всю необходимую информацию для записи его в файл. На этом месте стоит остановиться подробнее. Дело в том, что видеопоток в файле может храниться в сжатом виде, компрессированный соответствующим кодеком. Для того чтобы дать команду VFW сжимать поступающие картинки с помощью соответствующего кодека, нужно выполнить ряд определенных действий. Сначала нам нужно выполнить первичную инициализацию экземпляра структуры TAVICompressOptions. Данная структура предназначена для задания опций сжатия мультимедийных потоков (как видео, так и аудио). Следует отметить, что данная структура используется также и в функции AVISave, которую мы рассмотрим далее. Поэтому остановимся на ней подробнее.

Структура TAVICompressOptions имеет следующие поля, представленные в табл. 4.15.

Таблица 4.15. *Поля структуры* `TAVICompressOptions`

Поле	Описание
fccType	Идентификатор вида потока. Данное поле может принимать следующие значения: StreamtypeAUDIO — аудиопоток; StreamtypeVIDEO — видеопоток; StreamtypeMIDI — музыкальный MIDI-поток; StreamtypeTEXT — текст

Таблица 4.15 *(продолжение)*

Поле	Описание
fccHandler	Данное поле предназначено только для видеопотока. Поле задает кодер сжатия, указанный четырехбуквенным кодом (FourCC). Список идентификаторов кодеров представлен в табл. 4.7
dwKeyFrameEvery	Данное поле определяет максимальный период между ключевыми кадрами/блоками. Значение поля используется только при выставленном флаге AVICOMPRESSF_KEYFRAMES. Если данный флаг не выставлен, то каждый блок данных считается ключевым
dwQuality	Задает степень компрессии видеоданных. Поле может принимать значение от 0 до 1000. Для аудиопотоков не используется
dwBytesPerSecond	Скорость передачи. Значение задает количество байтов, выделенных для хранения 1 секунды потока, так называемый data rate. Значение данного поля используется только при выставленном флаге AVICOMPRESSF_DATARATE
dwFlags	Флаги компрессии: AVICOMPRESSF_INTERLEAVE — флаг указывает компрессору, что поток должен быть чересстрочным (interleaved) каждые dwInterleaveEvery кадров; AVICOMPRESSF_KEYFRAMES — флаг указывает компрессору, что этот видеопоток должен быть с ключевыми кадрами/блоками, следующими через определенное количество кадров/блоков, заданных в dwKeyFrameEvery; AVICOMPRESSF_DATARATE — указывает компрессору, что этот видеопоток должен быть сжат с заданной в параметре dwBytesPerSecond скоростью передачи; AVICOMPRESSF_VALID — очень интересный флаг, указывающий на то, что заполненная структура имеет корректные данные. Если этот флаг установлен, то AVIFile использует данные структуры для установки степени сжатия для функции AVISaveOptions. Если флаг не установлен, некоторые параметры кодера могут быть выставлены автоматически (по умолчанию)
lpFormat	Указатель на формат данных. Для аудиоданных указатель должен быть на структуру WAVEFORMATEX, а для видео — указатель на структуру BITMAPINFOHEADER
cbFormat	Размер структуры, указанной в lpFormat. Размер структуры можно получить с помощью функции SizeOf

Таблица 4.15 *(окончание)*

Поле	Описание
lpParms	Поле используется для внутренних целей и предназначено для хранения определенных компрессором данных. Программисту нет резона использовать данное поле
cbParms	Размер данных в lpParms
dwInterleaveEvery	Поле определяет, как часто чередовать данные потока с данными первого потока. Это поле используется компрессором при установленном флаге AVICOMPRESSF_INTERLEAVE

После того как структура TAVICompressOptions создана и инициализирована, можно приступать к работе со сжатым потоком. Но перед этим мы немного остановимся на функции AVISaveOptions. Хотя для дальнейшей работы данная функция не важна, но для практического применения очень даже полезна. Дело в том, что она позволяет вывести окно настройки и выбора кодера сжатия. То есть, используя данную функцию, мы можем предоставить пользователю самостоятельно выбрать нужный ему кодек с его собственными настройками сжатия. После выбора пользователем кодека программист может получить обновленную структуру TAVICompressOptions и продолжить работу.

Функция AVISaveOptions имеет следующий вид:

```
function AVISaveOptions(hParent: HWnd;
         uiFlags: Word; nStreams: Integer;
         var pStream : PAVIStream;
         var pacoOptions: PAVICompressOptions): WordBool
```

В качестве данных функция принимает следующие параметры, представленные в табл. 4.16.

Таблица 4.16. *Параметры* AVISaveOptions

Параметр	Описание
hParent	Хэндл родительского окна
uiFlags	Флаги, настраивающие диалоговое окно выбора сжатия. Определены следующие флаги: ICMF_CHOOSE_KEYFRAME — показывает количество ключевых кадров в диалоговом окне; ICMF_CHOOSE_DATARATE — показывает скорость передачи (data rate); ICMF_CHOOSE_PREVIEW — показывает окно предварительного просмотра

Таблица 4.16 *(окончание)*

Параметр	Описание
nStreams	Определяет количество потоков. Обычно выставляется значение, равное 1
pStream	Дескриптор потока
pacoOptions	Указатель на структуру TAVICompressOptions

На выходе функция вернет True, если в диалоговом окне будет нажата кнопка **ОК**. После завершения работы не забудьте освободить ресурсы, занятые AVISaveOptions, используя функцию AVISaveOptionsFree.

После установки опций сжатия мультимедийных потоков нужно получить указатель на сжатый поток, который, в свою очередь, создан от несжатого потока. То есть фактически нам нужно инициализировать сжатый поток и получить указатель на него. Делается это с помощью функции AVIMakeCompressedStream, имеющей вид:

```
function AVIMakeCompressedStream(var pCompressed: PAVIStream;
             pSource: PAVIStream;
             var acoOptions: TAVICompressOptions;
             var clsidHandler: TClsID): LongInt
```

Параметры функции представлены в табл. 4.17.

Таблица 4.17. *Параметры* *AVIMakeCompressedStream*

Параметр	Описание
pCompressed	Указатель на дескриптор сжатого потока
pSource	Дескриптор несжатого потока
acoOptions	Указатель на структуру TAVICompressOptions, предназначенную для установки опций сжатия
clsidHandler	Указатель на класс TClsid, используемый для создания потока

При успешном завершении функции переменная pCompressed будет указывать на дескриптор сжатого потока. Поток, в свою очередь, сможет записывать сжатые видеоданные в формате, заданном структурами TAVIStreamInfo и TAVICompressOptions.

Последним шагом перед непосредственной записью кадров является установка формата потока. Делается это с помощью функции AVIStreamSetFormat, имеющей следующий вид:

```
function AVIStreamSetFormat(pStream: PAVIStream;
             lPos: LongInt;
             lpFormat: LPVoid;
             cbFormat: LongInt): LongInt
```

Функция имеет следующие параметры, представленные в табл. 4.18.

Таблица 4.18. *Параметры* `AVIStreamSetFormat`

Параметр	Описание
`pStream`	Дескриптор потока. В нашем случае дескриптор компрессированного потока
`lPos`	Позиция в потоке. Так как в нашем случае формат потока будет одинаков в каждом кадре, устанавливаем значение, равное 0
`lpFormat`	Указатель на новый формат потока. Для видеоданных указатель должен быть установлен на структуру `BITMAPINFOHEADER`
`cbFormat`	Размер структуры, описывающей формат, заданный в `lpFormat`. Размер структуры можно узнать с помощью функции `SizeOf`

После выполнения всех этих функций мы можем начать запись кадров в поток. Запись кадров осуществляется с помощью функции `AVIStreamWrite`, имеющей следующий вид:

```
function AVIStreamWrite(pStream: PAVIStream;
            lStart,
            lSamples: LongInt;
            lpBuffer: LPVoid;
            cbBuffer: LongInt;
            dwFlags: DWord;
            plSampWritten,
            plBytesWritten: PLongInt): LongInt
```

Функция имеет следующие параметры, представленные в табл. 4.19.

Таблица 4.19. *Параметры* `AVIStreamWrite`

Параметр	Описание
`pStream`	Дескриптор потока. В нашем случае дескриптор компрессированного потока
`lStart`	Стартовый кадр, с которого пойдет запись
`lSamples`	Количество кадров для записи

Таблица 4.19 *(окончание)*

Параметр	Описание
lpBuffer	Указатель на буфер с данными для записи. При передаче видеокадра в объекте Bitmap нужно принудительно перевести битовое изображение в аппаратно-независимый растровый рисунок (DIB). Сделать это можно методом ScanLine класса Bitmap. То есть в параметре lpBuffer следует указать Bitmap.ScanLine[Bitmap.Height-1]
cbBuffer	Размер буфера с данными. При передаче видеокадра в объекте Bitmap указывается размер, равный Bitmap.Width × Bitmap.Height × кол-во байт, выделенных для хранения цвета. Если изображение представлено в 8-битном цвете, то умножить нужно на 1 байт, если в 16-битном — на 2, если в 24-битном — на 3 и если в 32-битном цвете, — соответственно, на 4
dwFlags	Флаг, указывающий состояние кадра/блока. Может принимать значение AVIIF_KEYFRAME, указывающее на то, что данный кадр/блок является ключевым
plSampWritten	Указатель на количество записанных кадров. По умолчанию лучше установить в Nil
plBytesWritten	Указатель на количество записанных байтов. По умолчанию лучше установить в Nil

Демонстрационный пример, записывающий файлы BMP в AVI-файл, вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch04\Ex04. На рис. 4.3 изображен результат просмотра сгенерированного файла AVI в плеере Light Alloy. Данный файл был сгенерирован с частотой 1 кадр в секунду.

Сокращенный пример функции, записывающей файлы BMP в AVI-файл, представлен в листинге 4.8.

Листинг 4.8. Пример записи файлов BMP в AVI

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
bmiHeader   : TBitmapInfoHeader;   // BMP-заголовок
AVIStreamInfo : TAVIStreamInfo;

CompOptions  : TAVICompressOptions;
PCompOptions : PAVICompressOptions;
```

```
clsidHandler : ^TClsID;

FName : String;
Bitmap : TBitmap;
begin
// Инициализация AVIFile.
AVIFileInit;
// Создаем файл
AVIFileOpen(tekFile, PChar(ExtractFilePath(Paramstr(0))+'\Demo.AVI'),
      OF_CREATE or OF_WRITE, nil);

// Обнуление
FillChar(bmiHeader,SizeOf(TBitmapInfoHeader),0);
with bmiHeader do // BITMAPINFOHEADER
begin
  biSize    := SizeOf(TBitmapInfoHeader);
  biWidth   := 480;           // Разрешение файла
  biHeight   := 352;
  biPlanes   := 1;            // Число плоскостей для
                        // целевого устройства.
  biBitCount  := 24;            // Число битов на пиксел.
  biCompression:= BI_RGB;          // Тип уплотнения
  // Размер одного кадра в пикселах
  biSizeImage :=
        (((biWidth * biBitCount) + 31) div 32) * 4 * biHeight;
end;

// Обнуление
FillChar(AVIStreamInfo,SizeOf(AVIStreamInfo),0);
// Заголовок создаваемого потока
with AVIStreamInfo do
begin
 dwSuggestedBufferSize:=bmiHeader.biSizeImage;
 fccType   := streamtypeVIDEO;
 fccHandler := GenerateFourCC('DIVX'); // FourCC идентификатор кодека
 dwFlags   := 0;
 dwScale   := 1;        // кадров в секунду = dwRate / dwScale
 dwRate   := SpinEditFrame.Value;
 dwStart   := 0;
 dwLength  := 1;
 dwSampleSize:= GenerateFourCC('CVID'); // Видеопоток
 dwSuggestedBufferSize:=bmiHeader.biSizeImage;
 // Прямоугольник для потока
 rectFrame.Right := bmiHeader.biWidth;
```

```
 rectFrame.Bottom := bmiHeader.biHeight;
end;

// Создаем поток
AVIFileCreateStream(TekFile,TekAVIStream,AVIStreamInfo);

// Подготавливаем информацию о компрессоре
// Обнуление
FillChar(CompOptions,SizeOf(CompOptions),0);
with CompOptions do // TAVICompressOptions
begin
 fccType      := AVIStreamInfo.fccType;  // Тип
 fccHandler    := AVIStreamInfo.fccHandler; // Кодер
 dwKeyFrameEvery  := SpinEditKeyFrames.Value; // Ключ.кадры
 dwQuality     := 7500;   //0-10000 качество сжатия
 dwFlags      := AVICOMPRESSF_KEYFRAMES or
           AVICOMPRESSF_DATARATE ;
 lpFormat     := @bmiHeader;
 cbFormat     := SizeOf(TBitmapInfoHeader);
 dwBytesPerSecond := 1024 * 100;  // Скорость передачи

 IF CheckBoxCompressOptions.Checked Then
 begin
   // Показываем диалог
   // Указатель на TAVICompressOptions
   PCompOptions  :=@CompOptions;
   dwFlags    :=dwFlags or AVICOMPRESSF_VALID;
   if not AVISaveOptions(Application.Handle,
              ICMF_CHOOSE_KEYFRAME or
              ICMF_CHOOSE_DATARATE,
              1,
              TekAVIStream,
              PCompOptions) then
   begin
    // Кодек не выбран
   end;
 end;
end;

// Обнуляем clsidHandler
clsidHandler:=nil;
// Первичная инициализация и создание сжатого потока
AVIMakeCompressedStream(CompStream,tekAVIStream,CompOptions,
            clsidHandler^);
```

```
// Устанавливаем настройки формата для потока
AVIStreamSetFormat(CompStream,0,@bmiHeader,SizeOf(TBitmapInfoHeader));

// Счетчик количества кадров
FramePos := 0;
// Записываем 3 кадра
Bitmap := TBitmap.Create;

FName := ExtractFilePath(Paramstr(0)) + '01.bmp';
Bitmap.LoadFromFile( FName );
WriteFrame(Bitmap);

FName := ExtractFilePath(Paramstr(0)) + '02.bmp';
Bitmap.LoadFromFile( FName );
WriteFrame(Bitmap);

FName := ExtractFilePath(Paramstr(0)) + '03.bmp';
Bitmap.LoadFromFile( FName );
WriteFrame(Bitmap);

Bitmap.Free;

IF CheckBoxCompressOptions.Checked Then
 // Освобождает ресурсы, связанные с AVISaveOptons
 AVISaveOptionsFree(1,PCompOptions);

// Закрытие потока и файла
AVIStreamRelease(CompStream);
AVIStreamRelease(tekAVIStream);
AVIFileRelease(TekFile);
// Закрытие API
AVIFileExit;

ShowMessage('файл AVI-файл сгенерирован !');
end;

function TForm1.WriteFrame(Bitmap: Graphics.TBitmap): Integer;
var
 Sze: integer;
begin
Bitmap.PixelFormat := pf24Bit;     // Глубина цвета
// В зависимости от глубины цвета выставляются коэффициенты
// В данном примере код оставлен для изучения, т. к. Bitmap.PixelFormat
// устанавливается равным 24 битам
case Bitmap.PixelFormat of
  pf8bit : Sze := 1;
```

```
  pf16bit: Sze := 2;
  pf24bit: Sze := 3;
  pf32bit: Sze := 4;
end;

Result:=AVIStreamWrite(CompStream,
            FramePos,
            1,
            Bitmap.ScanLine[Bitmap.Height-1],
                  // Прямой доступ к aDIB
            Bitmap.Width * Bitmap.Height * Sze, // Размер
            0,
            nil,
            nil);

// Увеличиваем счетчик кадров
Inc(FramePos);
end;
```

Рис. 4.3. Просмотр сгенерированного AVI-файла

Сохранение потоков в отдельных файлах

В данном разделе мы расскажем, какими способами можно создавать новый видео- или аудиофайл, загрузив в него поток из существующего AVI-файла, однако перед тем, как мы перейдем непосредственно к теме раздела, подытожим материал, представленный в предыдущих разделах. Хорошим примером

закрепления материала может служить небольшой демонстрационный проект, с предысторией которого вы сейчас познакомитесь.

В одном небольшом проекте нам понадобилось создать базу данных, которая хранила бы небольшие "слепки" видео, взятые с существующего видеоматериала. То есть требовалось создать нечто вроде базы данных предварительного просмотра (preview) накопленного видеоматериала. В нашем проекте предполагалось, что preview-кусок должен быть равен 1 минуте видео без звукового сопровождения. Используя материалы предыдущих разделов, вы сами легко сможете создать такой проект. Однако для того, чтобы упростить задачу, мы создали демонстрационный пример, сохраняющий выбранный кусок видеопотока в отдельный файл AVI.

На рис. 4.4 вы можете увидеть результат работы примера, вырезающего и сохраняющего кусок видео в отдельном AVI-файле.

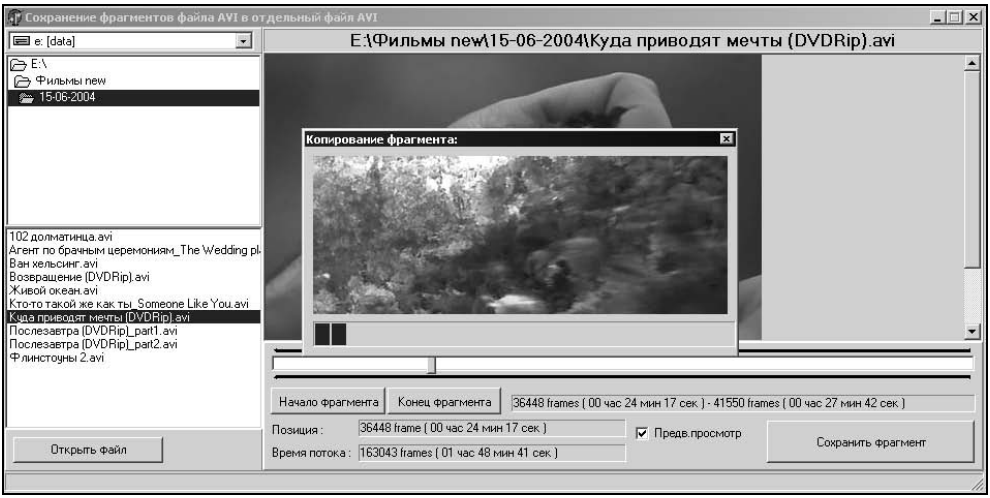


Рис. 4.4. Работа программы, вырезающей кусок видео

Исходный код примера вы сможете найти на прилагаемом компакт-диске в каталоге Source\Ch04\Ex05.

После небольшого отступления перейдем непосредственно к теме раздела. Несмотря на все недостатки приведенного примера, данный код вполне работоспособен, однако VFW предоставляет уже готовую функцию, позволяющую сохранить существующие потоки в отдельном файле.

Причем при выгрузке (записи) потоков вы можете задать и соответствующий компрессор для каждого потока в отдельности. То есть данная функция позволяет также выполнять пересжатие в другой формат существующих потоков, представленных в AVI-файле. Имя этой функции `AVISave`.

Данная функция имеет следующий вид:

```
function AVISave(szFile: PChar; pclsidHandler: PClsID;
        pfnCallback: Pointer; nStreams: Integer;
        pStream: PAVIStream;
        pacoOptions: PAVICompressOptions): LongInt
```

Входящие параметры функции `AVISave` представлены в табл. 4.20.

Таблица 4.20. *Параметры* `AVISave`

Параметр	Описание
`szFile`	Имя создаваемого файла
`pclsidHandler`	Указатель на дескриптор файла. При создании нового файла следует его установить в `Nil`

`pfnCallback`	Указатель на функцию обратного вызова `SaveCallBack`, которая должна иметь следующий вид:
	`Function SaveCallback(nPercent: Integer): Bool;`
	`pascal;`
	`begin`
	`// что-то здесь делаем`
	`end;`
	Данная функция делает сразу две операции. Во-первых, она позволяет получить и отобразить процесс выполнения операции сохранения потока. Достигается это за счет считывания переменной `nPercent` и передачи ее значения, например, в прогресс-индикатор.
	Во-вторых, она дает возможность пользователю прервать выполнение обработки в любой момент. Достигается это за счет обработки VFW результата работы функции. Если функция возвращает `True`, тогда VFW прекращает запись потока

`nStreams`	Количество сохраняемых (выгружаемых) потоков
`pStream`	Дескриптор потока-источника, сохраняемого (пересжимаемого) в файл. Перед вызовом функции данный поток должен быть уже открыт
`pacoOptions`	Указатель на структуру `TAVICompressOptions`

Замечание

Во время работы с данной функцией нами была выявлена очень неприятная ошибка. Дело в том, что в некоторых случаях после вызова функции работа ее завершается с ошибкой, и функция вместо `AVIERR_OK` возвращает

`AVIERR_FILEOPEN` (код `$8004406F`) — ошибка открытия файла. В результате долгого тестирования и не найдя устойчивой закономерности, мы пришли к выводу, что причина столь двоякого поведения функции, скорее всего, кроется в самой DLL. Обратите на это внимание.

В листинге 4.9 приведен сокращенный пример, выгружающий видеопоток открытого файла AVI в другой видеофайл. Полный исходный код примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch04\Ex06.

Листинг 4.9. Выгрузка видеопотока в другой файл AVI

```
function SaveCallback(nPercent: Integer): Bool; pascal;
begin
Application.ProcessMessages;
Form1.Progressbar.Position := nPercent; // Отображаем прогресс-бар
if abort = True then
  Result := True
else
  Result := False;
end;

procedure TForm1.BitBtnSaveClick(Sender: TObject);
var
CompOptions   : TAVICompressOptions;
PCompOptions   : PAVICompressOptions;
NewFile     : String;
Error      : Integer;
begin
Abort := False;
SaveDialog.FileName := 'Новый.avi';
IF SaveDialog.Execute Then
begin
 NewFile := SaveDialog.FileName;
 // Подготавливаем информацию о компрессоре
 FillChar(CompOptions,SizeOf(CompOptions),0);
 with CompOptions do // TAVICompressOptions
 begin
   // Показываем диалог
                 // Указатель на TAVICompressOptions
   PCompOptions  :=@CompOptions;
   dwFlags    :=dwFlags or AVICOMPRESSF_VALID;
   AVISaveOptions(Application.Handle,
          ICMF_CHOOSE_KEYFRAME or ICMF_CHOOSE_DATARATE,
          1,
```

```
          TekAVIStream,
          PCompOptions);

 end;
 BtnAbort.Visible := True;
 error := AVISave(Pchar(SaveDialog.FileName),nil,
          @SaveCallback,1,TekAVIStream, PCompOptions);
 BtnAbort.Visible := False;

 IF error <> 0 Then
   ShowMessage('Ошибка при вызове AVISave - Error code : ' +
         IntToHex(Error,4))
 else
   ShowMessage('Файл "'+ SaveDialog.FileName +
         '" успешно сохранен !');

  // Освобождает ресурсы, связанные с AVISaveOptons
  AVISaveOptionsFree(1,PCompOptions);
end;
end;

procedure TForm1.BtnAbortClick(Sender: TObject);
begin
 Abort := True; // Прервать сохранение
end;
```

Используя функцию `AVISave`, вы также можете выгружать и звуковую дорожку в звуковой файл. Для демонстрации мы переработали приведенный пример, добавив в него возможность выгрузки звуковой дорожки в отдельный файл. Полный исходный код примера вы найдете на прилагаемом компакт-диске в каталоге Source\Ch04\Ex07.

Как вы заметили, в предыдущих двух примерах функция `AVISave` выгружает только один поток: либо видео, либо аудио. Это обусловлено тем, что в данной версии обертки `AviFile32.pas` (а также и в `VFW.pas`) функция `AVISave` может выгружать только один поток, заданный в параметре `pStream`. На самом деле функция `AVISave` имеет возможность выгружать несколько потоков одновременно, например видео + аудио или видео + аудио + аудио. Достигается это за счет того, что функция может обрабатывать массивы, в которых заданы соответствующие потоки и настройки компрессоров. Так как в `AviFile32.pas` не предусмотрена такая возможность, мы пошли на хитрость и создали свою версию обертки функции `AVISave`, позволяющую выгружать сразу два потока одновременно (видео + аудио). Данную функцию мы назвали `AVISaveTwoStream`. Пример использования этой функции вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch04\Ex08.

В листинге 4.10 приведен пример определения данной функции как обертки функции AVISave (точнее, расширенного аналога AVISaveV).

Листинг 4.10. Собственная версия обертки функции AVISave

```
type
  ArrayPAVISTREAM          = array[0..1] of PAVISTREAM;
  ArrayPAVICompressOptions = array[0..1] of PAVICompressOptions;

function AVISaveTwoStream(szFile: PChar; pclsidHandler:
                          PCLSID; lpfnCallback: Pointer;
                          nStreams : integer;
                          pavi     : ArrayPAVISTREAM;
                          lpOptions: ArrayPAVICompressOptions): HResult;
                          stdcall;

function AVISaveTwoStream; external 'avifil32.dll' name 'AVISaveV'
```

Обработка ошибок VFW

Обработка ошибок — это одна из главных задач, которая встает перед программистами, желающими написать стабильно работающую программу. В данном разделе мы остановимся на обработке ошибок VFW.

Основным способом получения информации о произошедшей ошибке в VFW является анализ возвращаемого значения функций. Большинство функций VFW после выполнения возвращают числовой код, анализируя который можно узнать, с каким результатом завершила работу данная функция. В случае успешного выполнения функции VFW возвращают AVIERR_OK ($00000000). Если в работе функции произошла какая-то ошибка, VFW вернет соответствующий код ошибки, на основании которого программист может обработать возникшую ситуацию. Коды ошибок VFW и краткое описание приведены в табл. 4.21.

Таблица 4.21. *Коды ошибок VFW*

Ошибка	Код ошибки	Краткое описание
AVIERR_UNSUPPORTED	$80044065	Операция не может быть выполнена
AVIERR_BADFORMAT	$80044066	Ошибка в формате данных
AVIERR_MEMORY	$80044067	Нехватка памяти
AVIERR_INTERNAL	$80044068	Внутренняя ошибка

Таблица 4.21 *(продолжение)*

Ошибка	Код ошибки	Краткое описание
AVIERR_BADFLAGS	$80044069	Неверные флаги. Данная ошибка возникает, если функции были переданы неверные или взаимоисключающие флаги, принимаемые каким-либо параметром. Например, функции AVIFileOpen были переданы взаимоисключающие флаги, определяющие доступ к файлу
AVIERR_BADPARAM	$8004406A	Неверные параметры функции
AVIERR_BADSIZE	$8004406B	Ошибочный размер. Ошибка часто возникает, если программист неправильно определил размер структуры, которая в свою очередь передана в функцию в качестве параметра. Для корректного определения размера структур данных следует использовать SizeOf
AVIERR_BADHANDLE	$8004406C	Хендл, переданный в качестве параметра в функцию, неверен
AVIERR_FILEREAD	$8004406E	Ошибка чтения файла. Появление данной ошибки обычно связано с правами доступа к файлу
AVIERR_FILEWRITE	$8004406D	Ошибка записи в файл. Появление данной ошибки обычно связано с правами доступа к файлу
AVIERR_FILEOPEN	$8004406F	Ошибка открытия файла
AVIERR_COMPRESSOR	$80044070	Ошибка кодека. Причина появления данной ошибки довольно неопределенная, начиная от неверных данных и заканчивая неправильными установками параметров кодера
AVIERR_NOCOMPRESSOR	$80044071	Кодек не найден. Проверьте, установлен ли кодек в системе
AVIERR_READONLY	$80044072	Только для чтения
AVIERR_NODATA	$80044073	Нет данных. Функция VFW во время своей работы затребовала данные, которых не обнаружила. Ошибка часто проявляется при работе с буферами данных
AVIERR_BUFFERTOOSMALL	$80044074	Буфер, выделенный под данные, слишком мал
AVIERR_CANTCOMPRESS	$80044075	Ошибка в кодеке

Таблица 4.21 *(окончание)*

Ошибка	Код ошибки	Краткое описание
AVIERR_USERABORT	$800440C6	Операция VFW прервана пользователем
AVIERR_ERROR	$800440C7	Неопределенная ошибка

Заключение

В данной главе мы коротко представили вашему вниманию интерфейс для работы с видео — Video for Windows. Нами не ставилась задача подробно и в деталях рассказать обо всех возможностях данного интерфейса, да это и невозможно было бы сделать, учитывая объем главы. Поэтому вашему вниманию мы представили наиболее интересные, с точки зрения прикладного программирования, возможности VFW. Теперь дело за вами. Дерзайте.

ГЛАВА 5

GDI+ — графический интерфейс нового поколения

Введение в GDI+

GDI+ — это графическое ядро нового поколения, представленное Microsoft в своих новых операционных системах — Windows XP® и .NET Server®. Именно GDI+ является лицом данных операционных систем.

Итак, после десятка лет царствования интерфейса GDI появилась библиотека, призванная заменить устаревшую версию графического ядра и предоставляющая разработчикам все достоинства своего предшественника вкупе со множеством новых мощных возможностей.

Вот некоторые из них:

- градиентная заливка;
- поддержка прозрачности;
- режимы улучшения изображения с помощью методов сглаживания (антиалиасинг) и префильтрации растровых изображений;
- расширенный набор сплайнов (кривые Безье, сплайны, имитирующие поведение натянутой и изогнутой стальной проволоки);
- координатные преобразования, позволяющие осуществлять операции трансформации объектов GDI+;
- регионы с поддержкой координатных преобразований;
- расширенная работа с растрами;
- поддержка популярных форматов графических файлов BMP, GIF, TIFF, JPEG, Exif (расширение TIFF и JPEG для цифровых фотокамер), PNG, ICON, WMF и EMF;
- добавлен новый формат EMF+, который позволяет сохранить на диск и затем проиграть последовательность графических команд.

В Delphi до сих пор используется библиотека GDI, предоставляющая разработчику скудный (по сегодняшним меркам) набор функций для рисования, поэтому программистам часто приходится идти на хитрость при работе с графикой, используя DirectX, OpenGL или библиотеки, имеющие прямой доступ к видеопамяти.

Новое графическое ядро GDI+ призвано существенно упростить и стандартизовать создание графических офисных приложений.

Установка и требования к работе

Для начала давайте определимся: те, кто работает на операционных системах Windows XP и .NET Server могут не беспокоиться — GDI+ уже включена в эти системы.

Для пользователей других операционных систем придется скачать с официального сервера Microsoft по адресу:

http://www.microsoft.com/msdownload/platformsdk/sdkupdate/psdkredist.htm

файл gdiplus_dnld.exe (размером чуть более 1 Мбайт) или взять его с прилагаемого компакт-диска из каталога Install\GDI+ SDK. Начальное окно инсталлятора изображено на рис. 5.1.

В состав дистрибутива входит инструкция по установке и динамическая библиотека GdiPlus.dll, которую необходимо скопировать в системный каталог операционной системы.

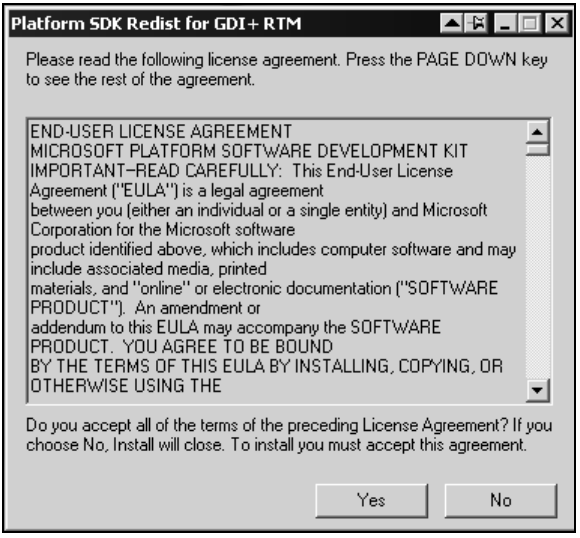


Рис. 5.1. Установка GDI+

Владельцам компьютеров, работающих под управлением Windows 95, придется с огорчением признать, что поддержка данной операционной системы не предусмотрена, и о ней нет (по крайней мере мы не нашли) никакого упоминания на сайте Microsoft. Теоретически можно предположить, что данная библиотека будет корректно работать и там.

Итак, библиотека успешно установлена и, предвидя следующий вопрос, добавим, для работы с GDI+ в Delphi вам потребуются заголовочные файлы, последние версии которых можно скачать с сайта open-source **http://www.progdigy.com** или взять с прилагаемого компакт-диска из каталога Install\GDI+ Headers.

Объектная модель библиотеки

GDI+ имеет объектную иерархию классов. Диаграмма самых необходимых классов библиотеки представлена на рис. 5.2.

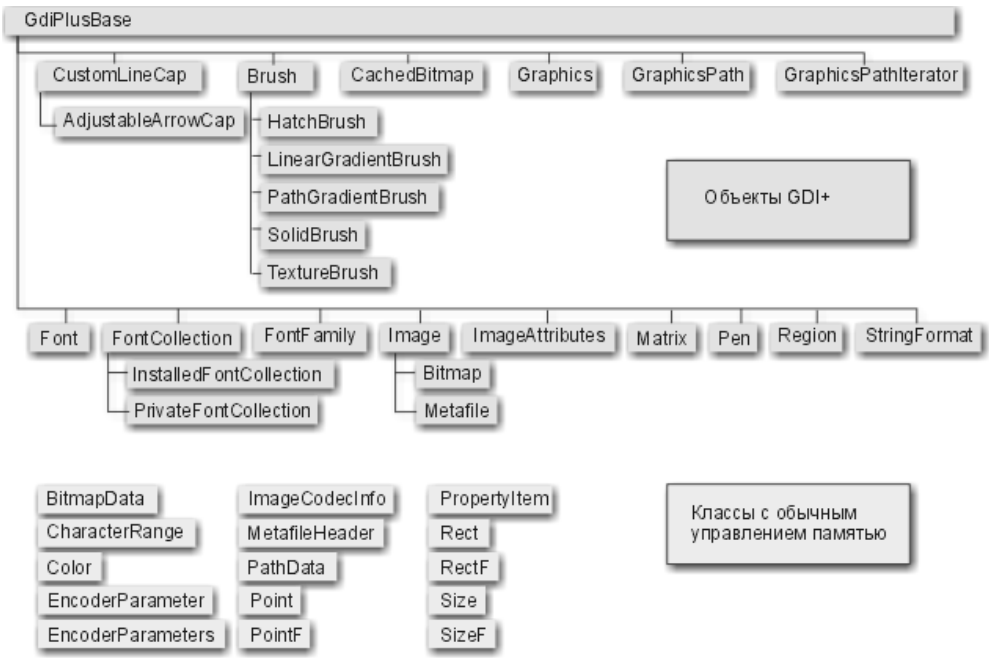


Рис. 5.2. Диаграмма классов GDI+

Как видим, большинство объектов имеют в корне иерархии класс `GdiPlusBase`. Вам не понадобится создавать экземпляры данного класса, т. к. он содержит только средства управления памятью библиотеки.

Все классы, инкапсулирующие работу с GDI+, порождены от материнского класса `GdiPlusBase`.

> ***Примечание***
>
> При использовании заголовочных файлов, взятых с сайта open-source **http://www.progdigy.com** или с прилагаемого к книге компакт-диска, к имени класса добавляется префикс `GP`, т. е. если класс GDI+ имеет название, например, `TSolidBrush`, то в Delphi он будет иметь название `TGPSolidBrush`. Это небольшое отличие состоит в том, что автор заголовков GDI+ внес разделение, чтобы не путать с созвучными классами, которые уже существуют в Delphi, как, например, `TFont`, `TPen`.

При работе с GDI+ на программиста ложится вся ответственность за создание, время жизни и удаление объектов. Это сильно отличается от привычного нам GDI в Delphi, где мы не заботимся о создании и удалении графических объектов, а просто используем их.

Первые шаги

После того как вы скачали и установили библиотеку GdiPlus.dll, а также получили заголовочные файлы, вы должны подключить эти файлы в секцию `uses` модуля вашей программы и объявить материнский класс `TGPGraphics`. Пример исходного кода представлен в листинге 5.1.

Листинг 5.1. Объявление материнского класса `TGPGrapchics`

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, ComCtrls, ExtCtrls, ToolWin,
  GDIPAPI,GDIPOBJ; // Для подключения GDI+ нужно
                   // подключить два модуля - GDIPAPI,GDIPOBJ;
type
  TForm1 = class(TForm)
    PaintBox1: TPaintBox;
    procedure PaintBox1Paint(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;
// Объявление класса TGPGraphics, ответственного за работу GDI :)
var
  graphicsGDIPlus : TGPGraphics;
```

Для первичного ознакомления с GDI+ мы решили положить на форму объект `PaintBox` и в обработчик события `OnPaint` объекта вывести текст "Hello GDI+" пятью различными стилями: обычным без сглаживания, обычным со сглаживанием, с градиентной заливкой, с текстурной заливкой и под углом 45°.

Для вывода графики в конструкторе объекта `TGPGraphics` требуется указать контекст устройства (DC) `Thandle`, куда библиотека будет направлять всю графику. После рисования нужно не забыть корректно освободить память.

Пример кода демонстрационного примера, выводящего "Hello GDI+", представлен в листинге 5.2.

Листинг 5.2. Hello GDI+

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
Const
  StrHello = 'Hello GDI+';
var
  R            : TRect;
  FontFamily   : TGPFontFamily;
  Font          : TGPFont;
  SolidBrush    : TGPSolidBrush;          // Заливка непрерывным цветом
  GradientBrush : TGPLinearGradientBrush; // Заливка линейным градиентом
  TextureBrush  : TGPTextureBrush;        // Заливка текстурой
  Image         : TGPImage;               // Объект - Изображение
  Matrix        : TGPMatrix;              // Матрицы
begin
  graphicsGDIPlus := TGPGraphics.Create(PaintBox1.Canvas.Handle);
  // Имя шрифта
  FontFamily := TGPFontFamily.Create('Times New Roman');
  // Шрифт
  Font       := TGPFont.Create(FontFamily, 32, FontStyleRegular,
                               UnitPixel);
  // Создаем объект для непрерывной заливки
  SolidBrush := TGPSolidBrush.Create(MakeColor(255, 0, 0, 255));
  // Рисуем текст без антиалиасинга (сглаживания) с закраской синим
  // цветом
```

```
  // Устанавливаем стиль отрисовки текста-
  // TextRenderingHintSingleBitPerPixel
graphicsGDIPlus.SetTextRenderingHint(TextRenderingHintSingleBitPerPixel);

  graphicsGDIPlus.DrawString(StrHello, -1, Font, MakePoint(1, 10.0),
solidBrush);

  // Рисование текста с антиалиасингом с закраской синим цветом
  // Установка стиля отрисовки текста - TextRenderingHintAntiAlias
  graphicsGDIPlus.SetTextRenderingHint(TextRenderingHintAntiAlias);
  graphicsGDIPlus.DrawString(StrHello, -1, Font, MakePoint(1, 40.0),
solidBrush);

  // Рисование текста с антиалиасингом с закраской градиентом
  R.X := 1;
  R.Y := 1;
  R.Width := 100;
  R.Height := 40;
  // Создаем объект для градиентной заливки
  GradientBrush := TGPLinearGradientBrush.Create(R,MakeColor(255, 255,
255, 255),MakeColor(255, 0, 0, 255),LinearGradientModeForwardDiagonal);
  graphicsGDIPlus.SetTextRenderingHint(TextRenderingHintAntiAlias);
  graphicsGDIPlus.DrawString(StrHello, -1, Font, MakePoint(1, 70.0),
GradientBrush);

  // Рисуем текст с антиалиасингом и с закраской текстурой
  // Создаем шрифт заново
  Font.Free;
  Font       := TGPFont.Create(FontFamily, 70, FontStyleRegular,
UnitPixel);
  Image  := TGPImage.Create('01.jpg');
  TextureBrush := TGPTextureBrush.Create(image);
  graphicsGDIPlus.SetTextRenderingHint(TextRenderingHintAntiAlias);
  graphicsGDIPlus.DrawString(StrHello, -1, Font, MakePoint(1, 100.0),
TextureBrush);

  // Рисуем под углом - используем трансформацию
  // Создаем шрифт заново
  Font.Free;
  Font       := TGPFont.Create(FontFamily, 32, FontStyleRegular,
UnitPixel);
  graphicsGDIPlus.RotateTransform(-45); // Производим
  graphicsGDIPlus.DrawString(StrHello, -1, Font, MakePoint(-200, 200.0),
TextureBrush);
  graphicsGDIPlus.ResetTransform;       // Сбрасываем
```

```
  // Не забываем высвободить память
  Image.Free;
  GradientBrush.Free;
  TextureBrush.Free;
  SolidBrush.Free;
  graphicsGDIPlus.Free;
end;
```

На рис. 5.3 вы видите результат работы нашего приложения.

Рис. 5.3. Результат вывода разными стилями "Hello GDI+"

После первичного знакомства с библиотекой рассмотрим детально работу с ней. Ознакомимся с классами и методами GDI+.

Классы GDI

В состав библиотеки входят следующие классы, представленные далее в алфавитном порядке.

Класс *AdjustableArrowCap*

Класс `AdjustableArrowCap` формирует форму концевых участков линии в виде стрелок.

Класс *Bitmap*

Данный класс предназначен для работы с растровыми изображениями. Класс обеспечивает возможности по созданию и управлению растрами. Данный класс наследуется от класса `Image`, который в свою очередь является базовым для работы с изображением: как с растровым, так и с векторным.

Класс `Bitmap` предоставляет возможность загружать изображение как из файла (причем при загрузке определяется формат изображения), так и из потоков. Приятным дополнительным новшеством является встроенная поддержка форматов JPG, TIF, GIF и PNG. Стоит также отметить, что в классе `Bitmap` реализована работа и с иконками (ICO).

Класс *BitmapData*

Класс `BitmapData` используется методами `LockBits` и `UnlockBits` класса `Bitmap` для получения прямого доступа к некоторой прямоугольной области растра.

Класс *Brush*

Родительский класс `Brush` предоставляет возможности по различной заливке сплошных областей и инкапсулирует соответствующие методы.

Сам по себе этот класс практически бесполезен, т. к. его нельзя использовать непосредственно для заливки. Все операции предоставлены в классах-потомках, таких как: `SolidBrush`, `LinearGradientBrush`, `HatchBrush`, `PathGradientBrush`, `TextureBrush`.

Класс *CachedBitmap*

Класс `CachedBitmap` представляет собой класс для работы с растрами, которые оптимизированы для вывода на конкретное устройство. Этот класс позволяет GDI+ наиболее быстро выводить растры. При создании данного класса необходимо указать оригинальный растр и устройство, на которое будет происходить вывод изображения. Скорость вывода растра накладывает и определенные ограничения. В частности, устройство вывода `Graphics` не должно быть принтером или метафайлом. Также при смене характеристик устройства вывода класс `CachedBitmap` необходимо пересоздавать заново, иначе вывод на устройство производиться не будет.

Класс *CharasterRange*

Класс, определяющий диапазон символьных позиций в пределах строки. Применяется при работе со шрифтами.

Класс *Color*

Класс `Color` предоставляет программисту возможность работы с цветом. Новшеством класса `Color` является полная поддержка 32-разрядного значения описания цвета. Теперь цвет можно представлять не только в виде цветовой модели `RGB`, как это принято в обычном `GDI`, но и добавить к этой модели величину непрозрачности — alpha, проще говоря, величину непрозрачности пиксела. Запись о цвете хранится в структуре `ARGB` типа `DWORD`. Информация о цвете представлена в виде:

- первые 8 битов — содержат Alpha-канал `a`;
- вторые 8 битов — содержат величину содержания красного цвета `r`;
- третьи 8 битов — содержат величину содержания зеленого цвета `g`;
- последние 8 битов — содержат величину содержания синего цвета `b`.

Следует также отметить, что оставлена и традиционная форма представления `Color` с тремя цветовыми компонентами, при этом значение `Alpha` устанавливается по умолчанию равным 255 (полная непрозрачность). Такая универсальность достигнута за счет перегрузки метода `MakeColor`.

Класс *CustomLineCap*

Класс, определяющий форму концевых участков линии, основанную на геометрическом пути. Концевой участок линии может иметь различную форму и представлять собой квадрат, круг, стрелку, значок алмаза и прочие фигуры. Для более сложных фигур, например стрелок, можно использовать готовый класс `AdjustableArrowCap`. Данный класс используется классом `Pen` при прорисовке линий и форм. Внутренние области концевых участков, представленные в виде замкнутых фигур (например, ромбов), в свою очередь могут быть залиты кистью `Brush`.

Интервал между значком и самой линией может быть откорректирован.

Класс *EncoderParametr*

Класс содержит параметры, которые передаются на кодер изображения при записи растра в определенный формат (JPG, PNG, TIFF и пр.). Класс может также использоваться для получения списка возможных значений, объявленных в соответствующем параметре кодера изображения. Так, например, вы можете с помощью данного класса добавить комментарий к TIFF-файлу или установить степень сжатия JPEG, т. е. данный класс как бы предоставляет систему команд для управления кодером при сохранении растра.

Класс *EncoderParametrs*

Данный класс предоставляет массив классов `EncoderParametrs` с функциями получения информации о количестве классов в массиве с помощью метода `Count` и возможностью доступа к конкретному классу `EncoderParametrs` через метод `Parameter`.

Класс *Font*

Данный класс инкапсулирует характеристики определенного шрифта, заданного в конструкторе класса, такие как высота, размер, стиль (или комбинация стилей).

Данный класс используется при прорисовке строк.

Класс *FontCollection*

Класс `FontCollection` является абстрактным классом и содержит методы для получения информации о шрифтах, находящихся в коллекции шрифтов. Объекты, сформированные из этого класса, включают классы `PrivateFontCollection` и `InstalledFontCollection`.

Класс *FontFamily*

Данный класс формирует набор шрифтов, которые составляют семейство одного шрифта. Семейство шрифта — это группа шрифтов, которые имеют один и тот же шрифт, но различаются, например, в начертании или толщине букв.

Класс *GDIPlusBase*

Базовый класс `GDIPlusBase` служит для распределения и освобождения памяти для графических устройств. Как было сказано прежде, вы никогда не должны создавать в своей программе экземпляр объекта `GDIPlusBase`.

Класс *Graphics*

Класс `Graphics` является ключевым в GDI+. Данный класс содержит почти пару десятков классов и пару сотен методов, отвечающих за рисование, отсечение и параметры устройства вывода. Прослеживается прямая связь со старым GDI, связанная с контекстом устройства (`Device Context`), эта связь видна при использовании двух конструкторов, которые создают экземпляр класса на основе хендла (HDC).

Главное же отличие по сравнению со старым GDI заключается в том, что теперь все методы GDI+ работают в собственных классах и не задействуют напрямую HDC. Достаточно один раз при создании класса `Graphics` указать контекст устройства вывода и все последующие методы рисования будут выводить результаты своей работы именно в это устройство.

Класс *GraphicsPath*

Интересный с точки зрения программиста класс `GraphicsPath` представляет собой как бы последовательность группировки графических команд, таких как вывод линий, окружностей, форм и пр. Этот класс удобно использовать при рисовании сложных объектов, которые в свою очередь подлежат трансформации. Объясним на примере. Предположим, вы нарисовали объект, состоящий из сотен линий и пары десятков окружностей ("деталь"). Вам нужно развернуть "деталь" на определенный градус. Вместо того, чтобы рисовать и рассчитывать новые координаты каждой линии и окружности после поворота (это трудоемко и требует знаний аналитической геометрии), вы с помощью данного класса как бы группируете все свои объекты в одну "деталь". Производите нужную трансформацию ее (в нашем случае это поворот) и, используя метод `DrawPath` объекта `Graphics`, выводите свою "деталь" в перевернутом виде, причем пересчет координат всех объектов, входящих в `GraphicsPath` после трансформации, GDI+ возьмет на себя. Вы можете объединять графические примитивы в разделы и работать с ними, используя класс `GraphicsPathIterator`.

Класс *GraphicsPathIterator*

Класс служит для работы с выбранными разделами пути, созданными в классе `GraphicsPath`. Стоит более подробно остановиться на данном классе. Дело в том, что, помимо общей группировки графических примитивов в классе `GraphicsPath`, этот же класс позволяет разбивать объекты, представленные графическими примитивами, на разделы с помощью маркеров. Используя приведенный пример с "деталью", можно было разбить нашу "деталь", например, на два раздела: линии и окружности. Причем раздел может состоять как из одного графического примитива, так и из множества. Создавать разделы в `GraphicsPathIterator` (устанавливать маркеры) можно с помощью метода `SetMarker` класса `GraphicsPath`.

Класс *HatchBrush*

Класс `HatchBrush` предоставляет средства заливки двухцветным узором на основе битового шаблона (`pattern`). GDI+ содержит уже 53 готовых шаблона, описываемых перечислением `HatchStyle`. Параметры `foreColor` и `backColor`

позволяют указать, соответственно, цвет линий шаблона и фоновый цвет кисти.

Класс *Image*

Данный класс является базовым для работы с различными изображениями (растровыми и векторными). Потомки данного класса — классы `Bitmap` и `Metafile` — собственно и реализуют требуемую функциональность при работе с растровыми и векторными изображениями.

Класс *ImageAttributes*

Для более гибкого контроля над выводом изображения в GDI+ создан класс `ImageAttributes`, который позволяет управлять цветовой коррекцией и геометрией вывода изображения (например, можно обеспечить вывод изображения мозаикой). Использование данного класса происходит при выводе изображения с помощью одного из перегруженных (overloaded) версий метода `DrawImage` объекта `Graphics`, где в одном из параметров передается указатель на объект класса `ImageAttributes`.

Класс *ImageCodecInfo*

Данный класс позволяет получить информацию о кодеке изображения библиотеки GDI+. У каждого кодека изображения имеется уникальный CLSID-идентификатор, позволяющий библиотеке точно идентифицировать соответствующий кодек. Чтобы получить список фильтров импорта кодеков, необходимо воспользоваться функцией `GetImageDecoders`, а для получения списка фильтров сохранения — функцией `GetImageEncoders`. Эти функции заполняют переданный им буфер размером `size` байт массивом структур `ImageCodecInfo`.

Несмотря на то, что в версии 1.0 библиотеки GDI+ кодеки не являются ActiveX-объектами как таковыми и встроены в саму библиотеку, а также отсутствует возможность работы с внешними кодеками, в будущих версиях Microsoft обещает переработать GDI+ и встроить в нее возможность поддержки кодеков независимых производителей.

Класс *InstalledFontCollection*

Класс `InstalledFontCollection` представляет шрифты, установленные в системе.

Примечание

Вы не должны использовать данный класс для установки новых шрифтов в систему. Если хотите установить шрифт, вы должны использовать функцию

AddFontResource, причем стоит отметить, что данный класс будет показывать только те шрифты, которые были в системе до установки нового шрифта.

Класс *LinearGradientBrush*

Назначение данного класса — заливка требуемой области цветовым переходом или градиентом цвета.

Класс *Matrix*

Класс Matrix представляет собой матрицу размером 3×3 и предназначен для осуществления операций поворота, переноса, масштабирования и отражения объектов GDI+.

Класс *Metafile*

Данный класс обеспечивает работу с векторными рисунками, так называемыми метафайлами. Метафайлы не привязаны к разрешению устройства вывода, для которого они создавались. Преимущество метафайлов видно при различных трансформациях векторных рисунков, таких как, например, масштабирование (при сильном увеличении изображения качество рисунка остается высоким). Разумеется, это не касается растровых изображений, присутствующих в метафайлах.

Класс *MetafileHeader*

Класс предназначен для получения информации о метафайле, например, об области векторного рисунка.

Класс *PathData*

Класс PathData является вспомогательным классом для классов GraphicsPathIterator и GraphicsPath. Класс GraphicsPath имеет массив точек и массив типов. Каждый элемент в массиве типов — байт, определяющий тип точки и набор флажков для соответствующего типа элемента в массиве точек. Вы можете использовать данный класс для получения и установки координат точек данных пути. Например, вы можете получить доступ к узловым точкам полилинии или кривой.

Класс *PathGradientBrush*

Назначение данного класса такое же, как и класса LinearGradientBrush, — заливка требуемой области цветовым переходом (градиентная заливка). От-

личием класса `PathGradientBrush` является возможность использования градиентного перехода сразу между несколькими цветами. Положение требуемых узловых точек определяется геометрическим путем (`path`), который может принимать произвольную форму.

Класс *Pen*

Класс `Pen` предназначен для настройки параметров "пера" различных выводимых графических примитивов, сплайнов и кривых Безье. Помимо знакомых свойств "пера" (цвет, толщина линии), в GDI+ добавлены многие другие свойства. Для линий стали доступны все возможные варианты градиентных заливок, текстур и шаблонов, различные геометрические характеристики линий. Например, вы можете нарисовать линию из набора параллельных линий различной толщины, а также задавать виды и формы концевых участков.

Класс *Point*

Класс `Point` полностью перешел из старого GDI и представляет собой класс хранения X и Y точки.

Класс *PointF*

Класс, идентичный классу `Point`, за исключением формата хранения координат. В данном классе координата точки хранится в вещественном типе.

Класс *PrivateFontCollection*

Класс `PrivateFontCollection` представляет собой собрание шрифтов. В собрание шрифтов попадают шрифты как установленные в системе, так и шрифты, которые не были установлены в системе, а используются как временные.

> ***Примечание***
>
> Данный класс позволяет разработчику использовать временные шрифты без установки их в систему, а также устанавливать частные версии шрифтов без замены шрифта в системе. Например, GDI+ может создать частную версию шрифта *Arial* в дополнение к системному шрифту *Arial*, который уже используется в системе.

Класс *PropertyItem*

Довольно необычный класс `PropertyItem` представляет программисту возможности работы с метаданными в изображениях. Не секрет, что в графических файлах можно скрыть дополнительные данные, и многие приложения

при записи изображений записывают туда различную информацию, например, авторские права, дату съемки или создания изображения. Вся эта дополнительная информация называется метаданными. Возможность хранения метаданных предусмотрена практически во всех графических форматах, поэтому в GDI+ и появился данный класс.

Класс *Rect*

Класс `Rect` представляет собой класс для хранения прямоугольной области.

Класс *RectF*

Класс, идентичный классу `Rect`, за исключением формата хранения данных о прямоугольной области. В этом классе данные хранятся в вещественном типе.

Класс *Region*

Очень интересный класс `Region` описывает область вывода для последующих графических команд. Различные изображения, графические примитивы, попадающие за пределы заданной области вывода, будут обрезаны. Область вывода может принимать любую форму и может быть комбинацией изогнутых и прямых линий, прямоугольников, графических путей.

Класс *Size*

Данный класс предназначен для хранения ширины и высоты 2-D-объекта в системе координат.

Класс *SizeF*

Класс, идентичный классу `Size`, за исключением формата хранения данных о размере 2-D-объекта. В этом классе данные хранятся в вещественном типе.

Класс *SolidBrush*

Самый простой класс из семейства `Brush`. Назначение класса — заливка областей однородным цветом. Существенное отличие класса от аналогичного графического объекта GDI — поддержка полупрозрачности.

Класс *StringFormat*

Данный класс инкапсулирует методы по форматированию выводимого текста (тип выравнивания, ориентация текста, вид обрезки текста при невозможно-

сти полностью вывести его в заданную область), а также предоставляет методы по управлению показом. Так, вы можете задать замену одного шрифта другим при невозможности показа символов, если эти символы не поддерживаются данным шрифтом. Класс используется в качестве параметра в одном из перегруженных методов `DrawString` класса `Graphics`.

Класс *TextureBrush*

Класс `TextureBrush` позволяет производить заливку растровым рисунком или текстурой. При инициализации объекта класса `TextureBrush` в конструкторе необходимо задать экземпляр класса `Image`. В качестве экземпляра класса `Image` могут выступать как растровые изображения (`Bitmap`), так и метафайлы (`Metafile`). Это, в свою очередь, предоставляет программисту огромные возможности: нарисовав замысловатый узор, вы можете проводить над ним различные трансформации и использовать этот узор для последующего рисования.

Перечисления GDI+

Помимо классов библиотека GDI+ содержит большое количество предопределенных *перечислений* (Enumerates) GDI+, использующихся в конструкторах и методах некоторых классов. Например, перечисление `DashStyle` определяет стиль рисования линий — рисование непрерывной линией; линией, состоящей из точек; линией штрих-точка-точка и другими стилями линий.

К сожалению, объем данной главы не позволяет подробно остановиться на перечислениях GDI+, поэтому в табл. 5.1 мы приводим лишь алфавитный список всех перечислений GDI+. Полную информацию по перечислениям вы можете получить в SDK из прилагаемого к книге компакт-диска.

Таблица 5.1. *Перечисления GDI+*

BrushType	DashCap	FlushIntention
ColorAdjustType	DashStyle	FontStyle
ColorChannelFlags	DriverStringOptions	HatchStyle
ColorMatrixFlags	EmfPlusRecordType	HotkeyPrefix
CombineMode	EmfType	ImageCodecFlags
CompositingMode	EncoderParameterValueType	ImageFlags
CompositingQuality	EncoderValue	ImageLockMode
CoordinateSpace	FillMode	ImageType

Таблица 5.1 *(окончание)*

InterpolationMode	PathPointType	StringDigitSubstitute
LinearGradientMode	PenAlignment	StringFormatFlags
LineCap	PenType	StringTrimming
LineJoin	PixelOffsetMode	TextRenderingHint
MatrixOrder	RotateFlipType	Unit
MetafileFrameUnit	SmoothingMode	WarpMode
MetafileType	Status	WrapMode
PaletteFlags	StringAlignment	

Константы и структуры GDI+

Помимо классов и перечислений GDI+ содержит огромное количество констант и восемь предопределенных структур. Например, структура `ColorPalette` определяет множество цветов, которые составляют цветовую палитру.

Практика использования

После краткого экскурса в GDI+ перейдем к практическим вопросам использования библиотеки в прикладных приложениях.

Рисование графических примитивов

Класс `Graphics` обеспечивает большое разнообразие методов по выводу графических примитивов. Основные методы работы с примитивами в GDI+ представлены в табл. 5.2.

Таблица 5.2. *Методы рисования графических примитивов GDI+*

Метод	Описание
`DrawArc`	Метод обеспечивает рисование сегмента эллипса (дуги). Эллипс определяется описывающим прямоугольником. Начальная точка сегмента `startAngle` и конечная `sweepAngle` задается в градусах. Сегмент рисуется против часовой стрелки
`DrawEllipse`	Метод обеспечивает рисование эллипса без заливки внутренней области эллипса. Эллипс вписан в задающий прямоугольник

Таблица 5.2 *(окончание)*

Метод	Описание
`DrawPie`	Метод обеспечивает рисование сектора эллипса, описываемого прямоугольником. Начальная точка сектора эллипса `startAngle` и конечная `sweepAngle` задается в градусах. Сегмент рисуется против часовой стрелки
`DrawPoligon`	Метод строит многоугольник, используя массив координат точек `TPoints`. Первая точка многоугольника соединяется с последней точкой. Заливка внутренней области не происходит
`DrawBezier`	Метод обеспечивает рисование кривой Безье (кубический сплайн). Для построения одной кривой Безье нужны минимум четыре точки — начальная, конечная и две опорные. По ним будет построена кривая второго порядка
`DrawBeziers`	Метод обеспечивает рисование групп кривых Безье. При задании массива точек они используются для построения последовательных кривых, причем последняя точка одной кривой является первой для следующей кривой
`DrawLine`	Метод обеспечивает рисование прямой линии
`DrawLines`	Метод обеспечивает рисование массива прямых линий, соединенных между собой (полилиний)
`DrawRectangle`	Метод обеспечивает рисование прямоугольника без заливки внутренней области
`DrawRectangles`	Метод обеспечивает рисование массива прямоугольников без заливки внутренних областей
`DrawPath`	Метод обеспечивает рисование графического пути (`Path`) без заливки внутренней области
`DrawCurve`	Метод обеспечивает рисование сглаженной кривой, проходящей по заданным точкам (сглаживание происходит с помощью кубических сплайнов). Точки, задающие прохождение кривой, передаются в массиве `TPoint`
`DrawClosedCurve`	Метод, аналогичный методу `DrawCurve` для рисования закрытой кривой

Следует заметить, что некоторые представленные в таблице методы, создающие замкнутые контуры, не производят заливку внутренних областей. Это объясняется тем, что в GDI+, в отличие от старого GDI, введена разбивка методов, производящих заливку и не производящих ее. Методы, производящие заливку графических примитивов, имеют префикс `Fill` и в остальном ничем не отличаются от таких же методов, имеющих префикс `Draw`. Напри-

мер, метод `DrawRectangle` рисует прямоугольник без заливки внутренней области, а метод `FillRectangle` рисует его с заливкой.

Ниже представлены методы, производящие заливку внутренних областей графических примитивов: `FillClosedCurve`, `FillEllipse`, `FillPath`, `FillPie`, `FillPolygon`, `FillRectangle`, `FillRectangles`.

На рис. 5.4 приведен результат использования данных методов по выводу разных графических примитивов.

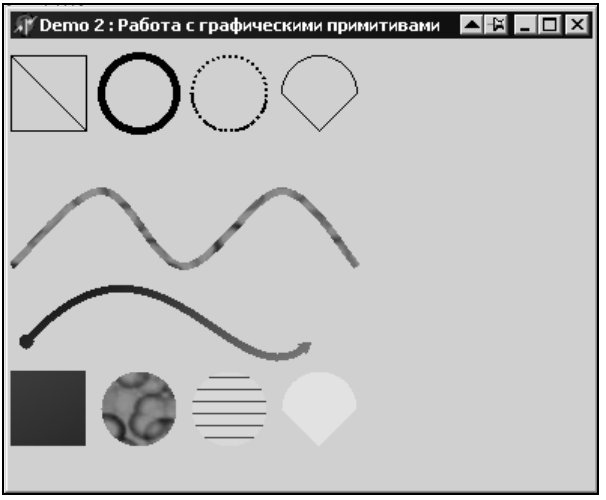


Рис. 5.4. Вывод графических примитивов

В листинге 5.3 представлены избранные места исходного текста данного примера, полный вариант вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\ Ex02.

Листинг 5.3. Вывод графических примитивов

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
Const
  dash : array[0..3] of single = (1, 1, 5, 4);
  points: array[0..4] of TGPPoint =
    ((x: 0  ; y: 150),     (x: 60 ; y: 100 ),     (x: 115; y: 150 ),
     (x: 180; y: 100 ),    (x: 230; y: 150));
var
  R             : TGPRect;
  Pen           : TGPPen;                  // Кисть
  path          : TGPGraphicsPath;         // Графический путь
```

```
  p1, c1, c2, p2: TGPPoint;
  Image         : TGPImage;                 // Изображение
  HatchBrush    : TGPHatchBrush;            // Кисть - двухцветный узор
                                            // На основе битового
                                            // шаблона
  TextureBrush  : TGPTextureBrush;          // Кисть - текстура
  GradientBrush : TGPLinearGradientBrush;   // Кисть - градиент
  SolidBrush    : TGPSolidBrush;            // Кисть - непрерывная
                                            // заливка
begin
  graphicsGDIPlus := TGPGraphics.Create(PaintBox1.Canvas.Handle);
  //-------------------------------------------------------------------
  // Рисование примитивов без заливки
  //-------------------------------------------------------------------
  // 2. Рисование эллипса (круга) с толщиной линии 5 пикселов
  pen:= TGPPen.Create(MakeColor(255, 0, 0, 0), 5);
  R.X := 60;  R.Y := 10;
  R.Width := 50;  R.Height := 50;
  graphicsGDIPlus.DrawEllipse(Pen,R);
  pen.Free;

  // 5. Рисование кривой (кубического сплайна) текстурой
  image := TGPImage.Create('01.jpg');
  TextureBrush := TGPTextureBrush.Create(image);
  Pen := TGPPen.Create(TextureBrush, 5); // В качестве пера
                                        // указываем текстуру
  graphicsGDIPlus.DrawCurve(pen, PGPPoint(@points), 5);
  pen.Free;
  image.Free;
  TextureBrush.Free;

  // 6. Рисование кривых Безье градиентной заливкой
  R.X := 10; R.Y := 150;
  R.Width := 190; R.Height := 100;
  GradientBrush := TGPLinearGradientBrush.Create(R,MakeColor(0, 0, 0),
                                                 MakeColor(0, 255, 0),
                                   LinearGradientModeForwardDiagonal);
  pen:= TGPPen.Create(GradientBrush, 5);
  pen.SetStartCap(LineCapRoundAnchor);
  pen.SetEndCap(LineCapArrowAnchor);
  p1 := MakePoint(10 , 200); // Начальная точка кривой
  c1 := MakePoint(100, 100); // Первая контрольная точка
  c2 := MakePoint(150, 250); // Вторая контрольная точка
  p2 := MakePoint(200, 200); // Конечная точка кривой
```

```
  graphicsGDIPlus.DrawBezier(pen, p1, c1, c2, p2);
  Pen.Free;
  GradientBrush.Free;
  //---------------------------------------------------------------------
  // Рисование примитивов с заливкой
  //---------------------------------------------------------------------
  // 1. Заливка градиентом
  GradientBrush := TGPLinearGradientBrush.Create(R,MakeColor(255, 0, 0),
                                                 MakeColor(0, 0, 255),
                                    LinearGradientModeForwardDiagonal);
  R.X := 0;  R.Y := 220;
  R.Width := 50;  R.Height := 50;
  // Рисуем прямоугольник
  graphicsGDIPlus.FillRectangle(GradientBrush,R);
  // Освобождаем "кисть"
  GradientBrush.Free;

  // 2. Заливка текстурой
  image := TGPImage.Create('01.jpg');
  TextureBrush := TGPTextureBrush.Create(image);
  R.X := 60;  R.Y := 220;
  R.Width := 50;  R.Height := 50;
  graphicsGDIPlus.FillEllipse(TextureBrush,R);
  image.Free;
  TextureBrush.Free;

  // Не забываем самое главное — освободить память
  graphicsGDIPlus.Free;
end;

end.
```

Работа с изображениями

Для работы с изображениями, как было ранее сказано, в GDI+ создан класс `Image`. Данный класс предоставляет программисту огромные возможности по работе с различными изображениями. Помимо работы с растрами и векторными изображениями, класс `Image`, например, позволяет создавать на основе изображения его мини-просмотр (`Thumbnail`), работать с графическими форматами, поддерживающими хранение нескольких картинок в одном файле, управлять цветовыми характеристиками и качеством изображения с помощью класса `ImageAttributes`.

Для вывода изображения существует всего один метод `DrawImage` класса `Graphics`, имеющий шестнадцать перегруженных версий вызовов и позволяющих по-разному выводить изображение и даже осуществлять его трансформацию, например, выводить изображение в виде развернутого прямоугольника или параллелограмма.

В основном варианты методов `DrawImage` делятся на две категории:

1. Методы, позволяющие выводить изображение или его части в прямоугольную область с возможностью масштабирования по двум координатным осям. Похожий эффект вывода изображения мы можем произвести, используя метод `StretchBlt` старого GDI, когда программист может вывести изображение, растянутое, например, в узкую полоску. В отличие от старого GDI, выводившего такие изображения с погрешностями прорисовки, новый GDI+ предоставляет более широкие возможности, в частности возможность интерполирования (сглаживания) изображения при выводе.
2. Методы, позволяющие выводить изображение или его части в параллелограмм. В старом GDI подобная функциональность отсутствовала вообще. Для работы этих методов в качестве параметров метода `DrawImage` требуется массив из 3-х точек, образующих вершины параллелограмма. Четвертая точка вычисляется автоматически.

Несмотря на такое многообразие версий метода `DrawImage`, некоторые простейшие трансформации изображения лучше проводить, используя координатные преобразования GDI+ или функции, предоставляющие подобную функциональность.

Простейший пример, производящий загрузку изображения из файла и вывод, представлен в листинге 5.4.

Листинг 5.4. Вывод изображения

```
 Image:= TGPImage.Create('Изображение.JPG');
// Вывод изображения как есть
graphicsGDIPlus.DrawImage(image, 0, 0);
Image.Free;
```

В данном примере изображение выводится в самом верхнем углу экрана. Еще раз обратим ваше внимание на то, что в экземпляре класса `TImage` можно создавать изображение и из векторного файла.

Далее мы рассмотрим интересные возможности GDI+, связанные с выводом графики.

Создание, запись и воспроизведение метафайлов

Метафайлы, или файлы с заданными последовательностями графических команд, появились еще в самой первой версии Windows. Концепция метафайлов так удачно зарекомендовала себя в последующих версиях, что мы уже не представляем себе отказ от данной технологии. Метафайлы распространились везде, начиная от офисных приложений и заканчивая CAD- и ГИС-системами. Действительно, выбранная концепция метафайлов очень проста — вы записываете графические команды в файл и в любой момент можете вывести их на любое устройство вывода. В отличие от растровых рисунков метафайлы не привязаны к разрешению устройства вывода, для которого они создавались, поэтому могут легко подвергаться трансформациям и масштабированию. Разумеется, это не касается растровых команд, которые также могут присутствовать в метафайлах.

Самые первые версии Windows поддерживали довольно ограниченный формат метафайлов — Windows Metafiles. Данные метафайлы поддерживали 16-битную координатную систему, не содержали информации о разрешении устройства вывода и позволяли записывать не все существующие команды GDI. Такие метафайлы существуют до сих пор и имеют расширение WMF.

По мере продолжения развития операционных систем вскоре оказалось, что и предложенный в Windows 95 формат EMF не удовлетворяет полностью потребностям разработчиков (например, EMF не сохраняет информацию о сглаживании графических примитивов). Для этого программисты Microsoft вынуждены были расширить оригинальный набор записей EMF. Такой расширенный формат получил название EMF+. Также для обратной совместимости был предложен дуальный формат (Dual EMF+), содержащий двойной набор записей EMF и EMF+. Файлы такого формата корректно открываются во всех операционных системах Microsoft, начиная с Windows 95.

В отличие от стандартного GDI, новый GDI+ предоставляет разработчику значительно больший выбор записываемых последовательностей графических команд, которые можно сохранить на диск в виде метафайла, хотя, при желании, в GDI+ может записывать и оригинальные EMF. При этом все вызовы графических команд GDI+ будут транслироваться в набор команд старого GDI.

В новом примере, представленном в листинге 5.5, мы переработали старый пример по выводу графических примитивов. В новом примере вывод происходит в метафайл с именем Sample.wmf, потом данный метафайл воспроизводится на экран с помощью метода `DrawImage` объекта `Graphics`.

Данный пример приведен частично, полную версию примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\ Ex03.

Листинг 5.5. Работа с метафайлом

```
// Создаем метафайл, куда будем выводить графические команды
metafile := TGPMetaFile.Create('Sample.emf');
// Создаем объект Graphics для вывода в метафайл
graphicsGDIPlus := TGPGraphics.Create(metafile);
//---------------------------------------------------------------------
// ПРОИЗВОДИМ ВЫВОД
//---------------------------------------------------------------------
// Создаем класс "кисть"
pen:= TGPPen.Create(MakeColor(255, 0, 0, 0), 1);
R.X := 0;  R.Y := 10;
R.Width := 50;  R.Height := 50;
// Рисуем прямоугольник
graphicsGDIPlus.DrawRectangle(Pen,R);
// Рисуем линию
graphicsGDIPlus.DrawLine(Pen,R.X,R.Y,R.X+R.Width,R.Y+R.Height);
// Освобождаем "кисть"
pen.Free;

// Не забываем самое главное - освободить память
graphicsGDIPlus.Free;

// Проигрываем метафайл
graphicsGDIPlus := TGPGraphics.Create(PaintBox1.Canvas.Handle);
graphicsGDIPlus.DrawImage(metafile, 0, 0);
graphicsGDIPlus.Free;
```

Обрезка и масштабирование изображений

Класс `Graphics` обеспечивает несколько вариантов методов `DrawImage`, которые вы можете использовать для обрезки и масштабирования изображений. Действие этих методов лучше рассмотреть на примере исходного кода, представленного в листинге 5.6.

Листинг 5.6. Обрезка и масштабирование изображений

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  Image          : TGPImage;                // Изображение
  width, height        : UINT;              // Размер
  NewWidth, Newheight  : UINT;              // Новый размер
  destinationRect: TGPRectF;                // Область
```

```
  WidthCrop       :UINT;                    // Ширина обрезаемого куска
  HeightCrop      :UINT;                    // Высота обрезаемого куска
  OffsetWidthCrop :UINT;                    // Смещение по ширине
  OffsetHeightCrop:UINT;                    // Смещение по высоте
begin
  // Создаем объект Graphics
  graphicsGDIPlus := TGPGraphics.Create(PaintBox1.Canvas.Handle);
  Image:= TGPImage.Create('Истребитель.JPG');
  width  := image.GetWidth;
  height := image.GetHeight;

  // Рисуем оригинальную картинку
  graphicsGDIPlus.DrawImage(image, 0, 0);

  // Выводим это изображение с 20 процентным уменьшением
  NewWidth := Round(width / 1.2);
  NewHeight:= Round(height/ 1.2);
  destinationRect := MakeRect(0,height, NewWidth, NewHeight);
  graphicsGDIPlus.DrawImage(
    image,
    destinationRect,  // Область вывода
    0, 0,             // Левый верхний угол исходного прямоугольника
    width,            // Размер по ширине исходного прямоугольника
    height,           // Размер по высоте исходного прямоугольника
    UnitPixel);

  // Выводим кусок изображения (делаем обрезку)
  OffsetWidthCrop := 250;
  OffsetHeightCrop:= 0;
  WidthCrop := 300;
  HeightCrop:= 200;

  // Обратите внимание, что размеры области вывода по длине и ширине
  // равны размерам обрезаемого исходного прямоугольника.
  destinationRect := MakeRect(0,height + NewHeight, WidthCrop,
HeightCrop);
  graphicsGDIPlus.DrawImage(
    image,
    destinationRect,  // Область вывода
    OffsetWidthCrop,OffsetHeightCrop,   // Смещение от левого верхнего
                                        // угла исходного прямоугольника
    WidthCrop,        // Размер по ширине исходного прямоугольника
    HeightCrop,       // Размер по высоте исходного прямоугольника
    UnitPixel);
```

```
  Image.Free;
  graphicsGDIPlus.Free;
end;
```

Полную версию примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\ Ex04.

На рис. 5.5 приведена схема, показывающая, как правильно определить область обрезки. Результат работы программы вы можете увидеть на рис. 5.6.

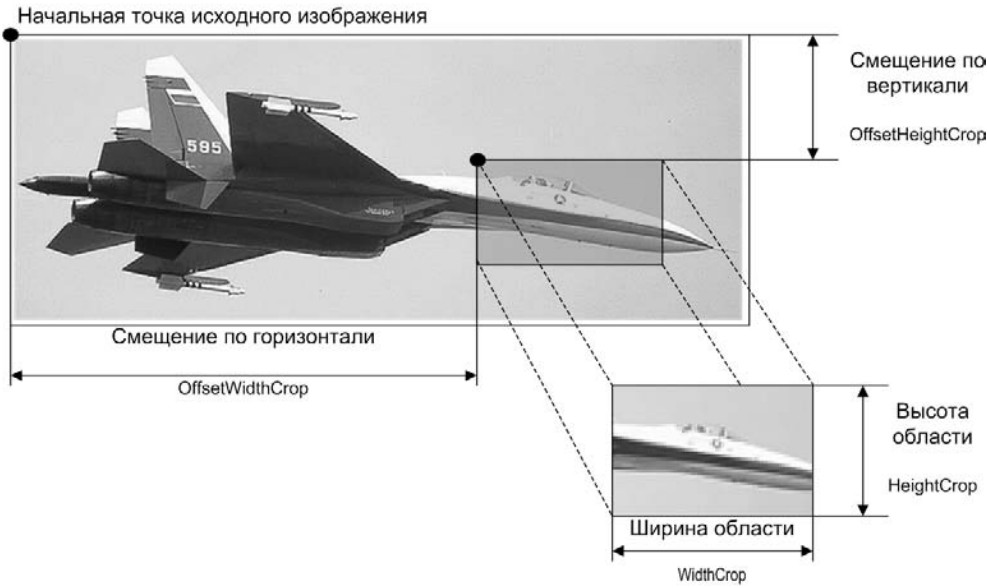


Рис. 5.5. Схема обрезки

Улучшение качества при выводе изображений и графических примитивов

В предыдущем разделе мы рассмотрели пример масштабирования изображения. Данный пример удовлетворяет практически все нужды прикладных программистов, однако у него есть существенный минус — при сильном масштабировании на изображении появляются "зубчики", связанные с искажениями растра. Для решения этой проблемы в GDI+ встроены средства повышения качества выводимого изображения — возможности интерполирования растра и антиалиасинг графических примитивов.

В листинге 5.7 представлен кусок примера программы, включающий в GDI+ режим бикубической интерполяции. Полную версию примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex05.

Рис. 5.6. Пример масштабирования и обрезки рисунков

Листинг 5.7. Использование бикубической интерполяции при выводе изображений

```
// Используем интерполяцию высокого качества (бикубическую)
graphicsGDIPlus.SetInterpolationMode
                                (InterpolationModeHighQualityBicubic);
graphicsGDIPlus.DrawImage(
  image,
  MakeRect((0.9*width)*2 + 1,Height, 0.9*width, 0.9*height),
                                                  // Область вывода
  0, 0,         // Смещение от левого верхнего угла
                // исходного прямоугольника
```

```
    width,       // Размер по ширине исходного прямоугольника
    height,      // Размер по высоте исходного прямоугольника
    UnitPixel);
```

Как видите, для установки режима интерполяции нужно воспользоваться методом `SetInterpolationMode` класса `Graphics`. Метод `SetInterpolationMode` в качестве параметров принимает элемент перечисления `InterpolationMode`. Варианты элементов метода представлены в табл. 5.3.

Таблица 5.3. *Параметры* `SetInterpolationMode`

`InterpolationModeDefault`	`InterpolationModeBicubic`
`InterpolationModeLowQuality`	`InterpolationModeNearestNeighbor`
`InterpolationModeHighQuality`	`InterpolationModeHighQualityBilinear`
`InterpolationModeBilinear`	`InterpolationModeHighQualityBicubic`

Примечание

К сожалению, в GDI+ нет нового перспективного фильтра Lancroz3, предоставляющего самые высокие результаты по интерполированию. Однако по заверениям разработчиков поддержка данного фильтра будет встроена в новые редакции GDI+.

На рис. 5.7 представлен вывод изображения без интерполяции в оригинальном виде, а на рис. 5.8 — тот же рисунок, выведенный с использованием бикубической интерполяции при уменьшении оригинала на 10%.

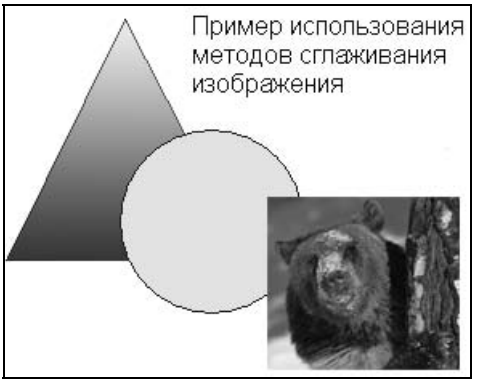


Рис. 5.7. Оригинал до масштабирования

Рис. 5.8. Изображение после интерполяции и масштабирования

Для включения режима антиалиасинга графических примитивов в GDI+ существует метод `SetSmoothingMode` класса `Graphics`. В качестве параметра ме-

тода используется элемент перечисления SmoothingMode. Варианты элементов SmoothingMode представлены в табл. 5.4.

Таблица 5.4. *Параметры* SmoothingMode

SmoothingModeDefault	SmoothingModeNone
SmoothingModeHighSpeed	SmoothingModeAntiAlias
SmoothingModeHighQuality	SmoothingModeInvalid

В данный момент в GDI+ не существует разницы между режимами SmoothingModeDefault, SmoothingModeHighSpeed и SmoothingModeNone — все они выключают антиалиасинг примитивов. Изображение при этом приобретает привычные зубчатые края.

Для включения режима сглаживания примитивов нужно задействовать константы SmoothingModeHighQuality или SmoothingModeAntiAlias.

Установка режима SmoothingModeInvalid вернет ошибку времени выполнения.

Пример по сглаживанию графических примитивов вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\ Ex06.

Повороты, наклоны и отражения изображений

Одним из несомненных достоинств GDI+ стала простая поддержка возможностей поворотов, наклонов и отражений исходного изображения. Данная функциональность была присуще ранее, пожалуй, только графическим редакторам. Теперь с помощью пары строк кода вы можете создавать отражения по вертикали и горизонтали, наклонять изображение различными способами, а также выводить его в параллелограмм.

Все это многообразие достигается вызовом всего лишь одной версии перегруженного метода DrawImage. В качестве параметра вызова в данный метод передается массив, состоящий из трех элементов TGPPoint, который, в свою очередь, определяет координаты точек области вывода. Координаты четвертой точки области вывода рассчитываются автоматически на основе переданных трех.

Очередность заполнения массива элементов определяется следующим образом: X,Y левого верхнего угла, X,Y правого верхнего угла и в качестве последнего параметра массива — X,Y левого нижнего угла изображения.

Примечание

Хочется сразу заметить, что для простых трансформаций, например, поворота изображения на 45°, лучше всего использовать методы трансформации GDI+. При использовании данных методов у вас отпадет необходимость собственно-

ручно рассчитывать координаты области вывода, библиотека GDI+ автоматически сделает это за вас. Методы трансформации будут рассмотрены ниже, данный же код лучше всего использовать для создания наклонов и отражений.

На рис. 5.9 приведена схема, показывающая, как правильно определить область вывода изображения при операциях поворота, наклона и отражения.

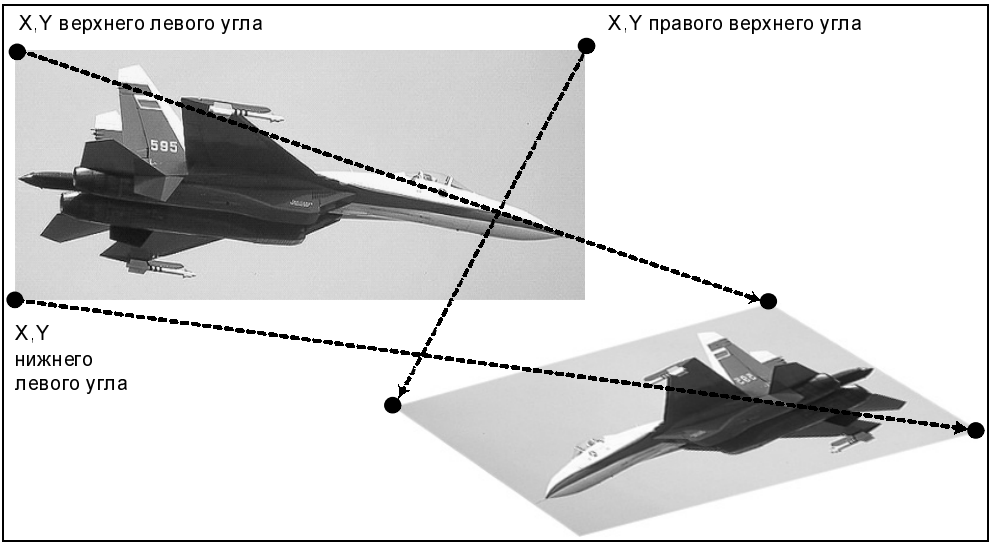


Рис. 5.9. Схема определения области вывода

В листинге 5.8 представлен фрагмент исходного кода, выводящий перевернутое по горизонтали исходное изображение истребителя. Полный исходный код примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\ Ex07.

Листинг 5.8. Перевороты и повороты изображения

```
Image:= TGPImage.Create('Истребитель.JPG');
widthOrig  := image.GetWidth;
heightOrig := image.GetHeight;
// Рисуем оригинальную картинку
graphicsGDIPlus.DrawImage(image, 0, 0);
// Выводим изображение, перевернутое по горизонтали
// Верхний левый угол выводимого изображения
destPoints[0].X := widthOrig * 2;
destPoints[0].Y := 0;
```

```
// Правый левый угол выводимого изображения
destPoints[1].X := widthOrig;
destPoints[1].Y := 0;
// Нижний левый угол выводимого изображения
destPoints[2].X := widthOrig * 2;
destPoints[2].Y := HeightOrig;
// Вывод
graphicsGDIPlus.DrawImage(image, PGPPoint(@destPoints), 3);
```

Результат работы программы представлен на рис. 5.10.

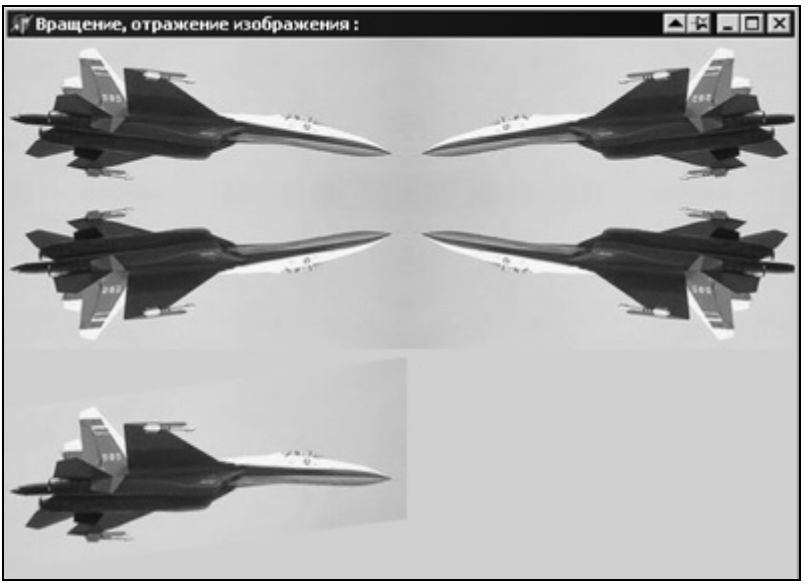


Рис. 5.10. Вращение и отражение изображения

Создание эскизов (мини-просмотров) изображения

Казалось бы, методы масштабирования уже позволяют выводить изображения в малых размерах и присутствие в GDI+ отдельного механизма для создания эскизов изображений (`thumbnail`) необязательно. Однако в большинстве графических форматов уже предусмотрено хранение эскиза вместе с изображением и нет смысла загружать и обрабатывать изображение, когда можно просто "вытащить" готовый эскиз. Поэтому метод `GetThumbnailImage` класса `Image` является наиболее быстрым способом получения эскизов.

Метод `GetThumbnailImage` имеет всего два параметра, задающих ширину и высоту создаваемого эскиза, и параметры, влияющие на возможность прерыва-

ния процесса создания эскиза. Параметры `thumbWidth` и `thumbHeight` задают размеры, а параметры `callback` и `callbackData` управляют возможностью прерывания; по умолчанию `callback` и `callbackData` равны `NULL`, и эта возможность игнорируется. Если же передать в параметрах `thumbWidth` и `thumbHeight` значение 0, GDI+ постарается извлечь готовый эскиз, хранящийся в изображении. Естественно при использовании неподходящего формата вам придется создавать эскиз самостоятельно.

Полный исходный код примера, создающего эскизы изображений файлов JPG, вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\ Ex08.

Результат работы программы представлен на рис. 5.11.

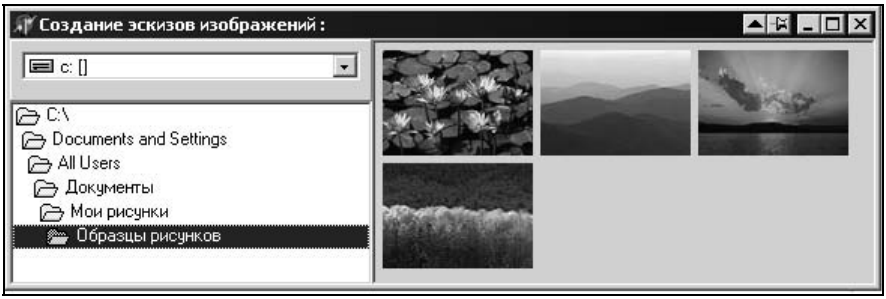


Рис. 5.11. Просмотр эскизов изображений

Использование кэшированных растров для повышения производительности вывода

Механизм хранения рисунков GDI+ по умолчанию подразумевает хранение изображения в независимом от устройства вывода формате. Такой вариант хранения предусматривает универсальность при выводе растра на различные устройства вывода (экран, принтер), т. к. в этом случае GDI+ автоматический производит конвертирование изображения, основанное на характеристиках данного устройства. Однако такой метод имеет и свои недостатки, пожалуй, самым важным из которых является низкая производительность при выводе. Для повышения производительности вывода и был создан зависимый от устройства вывода формат хранения растров, представленный в классе `CachedBitmap`. Для работы с ним нужно при создании экземпляра класса указать оригинальный растр и устройство, на которое будет происходить вывод изображения.

Так в примере, представленном в листинге 5.9, в начале пятьсот раз выводится некэшированный растр и в конце столько же раз производится вывод

кэшированного растра. После каждого вывода подсчитывается время, затраченное GDI+ на прорисовку. Результаты тестирования помещаются в нижнюю панель.

Листинг 5.9. Оптимизация вывода растров

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  bitmap    : TGPBitmap;                // Класс для хранения
                                        // устройство-независимого растра
  width, height: Integer;
  cBitmap: TGPCachedBitmap;             // Класс для хранения кэшируемого
                                        // (оптимизируемого) растра
  j: integer;
  T1, T2: TDateTime;
  h,m,s, ms: word;
begin
  // Создаем объект Graphics
  graphicsGDIPlus := TGPGraphics.Create(PaintBox1.Canvas.Handle);
  // Загружаем растр
  // Данный растр является устройство-независимым (неоптимизированным)
  bitmap := TGPBitmap.Create('Истребитель.JPG');
  width  := bitmap.GetWidth;
  height := bitmap.GetHeight;
  // Создаем устройство-зависимый (оптимизированный) растр
  cBitmap:= TGPCachedBitmap.Create(bitmap, graphicsGDIPlus);

  // Рисование неоптимизированного растра
  j := 0;
  T1 := now;
  While j < 500 do
  begin
    graphicsGDIPlus.DrawImage(bitmap, 0, j div 2, width, height);
    inc(j,1)
  end;
  T2 := now;
  DecodeTime(T2-T1,h,m,s,ms);
  Label3.Caption :=IntTostr(s) +':'+ IntTostr(ms);
  // Рисование оптимизированного растра
  j := 0;
  T1 := now;
  While j < 500 do
  begin
```

```
    graphicsGDIPlus.DrawCachedBitmap(cbitmap, Width, j div 2);
    inc(j,1);
  end;
  T2 := now;
  DecodeTime(T2-T1,h,m,s,ms);
  Label4.Caption :=IntTostr(s) +':'+ IntTostr(ms);

  bitmap.Free;
  cBitmap.Free;
  graphicsGDIPlus.Free;
end;
```

Полный код примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\ Ex09.

Использование кодеров и декодеров изображений

Для поддержки популярных графических форматов изображений в GDI+ был создан механизм кодеров/декодеров (компрессоров/декомпрессоров) изображений.

> ***Примечание***
>
> Кодер — это внутренний механизм, с помощью которого GDI+ может сохранить изображение в одном из представленных форматов: BMP, GIF, TIFF, JPEG, Exif (расширение TIFF и JPEG для цифровых фотокамер), PNG, ICON, WMF и EMF. Декодер, в свою очередь, выполняет противоположную задачу — позволяет GDI+ прочитать формат изображения и соответственно отобразить его. Также следует заметить, что декодеры выполнены с учетом специфики графических форматов.

Работа со списком кодеков

У каждого кодера и декодера GDI+ имеется уникальный CLSID-идентификатор, позволяющий GDI+ разделять эти кодеры и декодеры. Для получения списка кодеров и декодеров необходимо воспользоваться функциями: `GetImageDecoders` (декодеры) и `GetImageEncoders` (кодеры). Эти функции заполняют переданный им буфер размером `size` байт массивом структур `ImageCodecInfo`. Перед вызовом данных функций вы должны выделить буфер в памяти. Для правильного определения размера буфера вы должны воспользоваться функцией `GetImageEncodersSize`.

Для просмотра свойств установленных в системе кодеков GDI+ мы создали программу, полный код которой вы можете найти на прилагаемом компакт-

диске в каталоге Source\Ch05\ Ex10. Результат работы программы представлен на рис. 5.12.

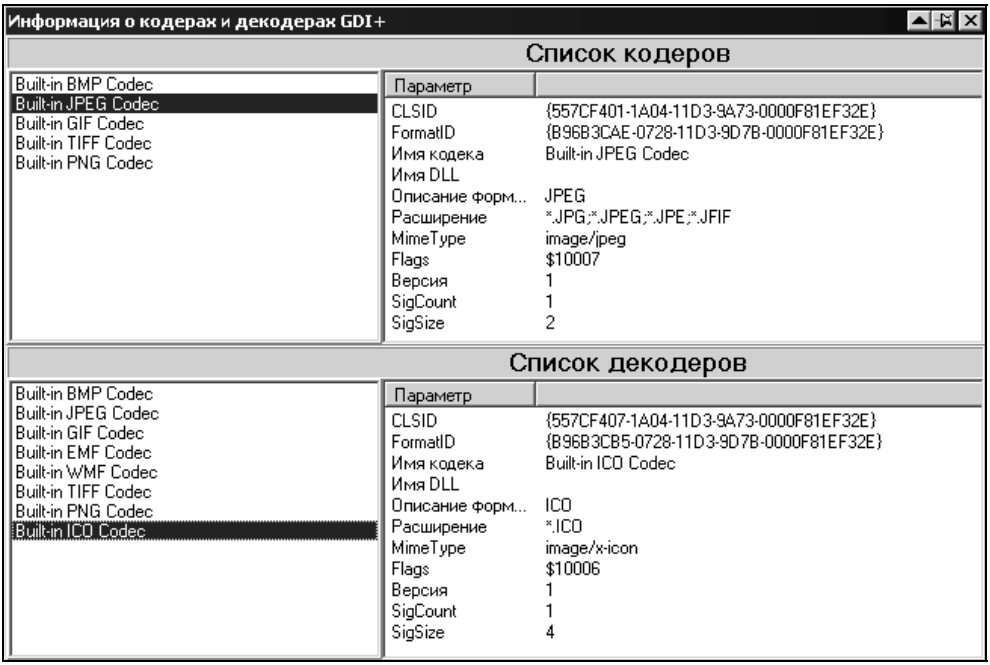


Рис. 5.12. Информация о кодерах и декодерах

Информация о свойствах кодеров нужна при различных операциях GDI+, например, при сохранении изображения. В качестве параметра требуется указать, кодек какого формата необходимо использовать. Для указания кодека нам нужно передать указатель на GUID кодека (CLSID).

Получение CLSID кодера изображения

Как было замечено, для сохранения изображения GDI+ необходим GUID выбранного кодека. Для программиста закономерно встает вопрос о том, где можно взять его значение? Самый простой способ — жестко завести в код константу, взяв ее, например, из окна приведенной программы, однако данный подход имеет существенный недостаток. Дело в том, что GUID-индентификатор в будущем при выходе новых версий может измениться, и тогда ваша программа потеряет функциональность. Для предотвращения этого Microsoft рекомендует более гибкий способ.

Данный способ заключается в том, чтоб произвести поиск GUID-кодека, основываясь на значении поля `MimeType` объекта `ImageCodecInfo`, и при успеш-

ном завершении поиска вернуть его. Так, скажем, для получения GUID-кодека JPEG формируем поисковую строку вида "image/jpeg" и производим поиск. Реализация этого подхода имеется в документации — в разделе "Retrieving the Class Identifier for an Encoder", а в листинге 5.10 представлен исходный код функции `GetEncoderClsid`, приведенный на Object Pascal.

Данная функция находится в файле GDIPUTIL.pas заголовочных файлов GDI+.

Листинг 5.10. Использование функции `GetEncoderClsid`

```
function GetEncoderClsid(format: String; out pClsid: TGUID): integer;
var
num, size, j: UINT;
ImageCodecInfo: PImageCodecInfo;
Type
ArrIMgInf = array of TImageCodecInfo;
begin
num := 0;
size := 0;
result := -1;
GetImageEncodersSize(num, size);
if (size = 0) then exit;
GetMem(ImageCodecInfo, size);
if(ImageCodecInfo = nil) then exit;
GetImageEncoders(num, size, ImageCodecInfo);
for j := 0 to num - 1 do
begin
 if( ArrIMgInf(ImageCodecInfo)[j].MimeType = format) then
 begin
  pClsid := ArrIMgInf(ImageCodecInfo)[j].Clsid;
  result := j; // Success
 end;
end;
FreeMem(ImageCodecInfo, size);
end;
```

Определение параметров кодера

При записи изображения для некоторых кодеков вы можете задать дополнительные настройки записи, например, установить степень сжатия изображения или глубину цвета. Данная функциональность достигается за счет дополнительных параметров, предоставляемых кодером записи. Для получения

этих параметров в классе `Image` предусмотрен метод `GetEncoderParameterList`, заполняющий буфер размером `size` байт массивом структур `EncoderParameter`. Перед вызовом `GetEncoderParameterList` вы должны выделить буфер в памяти. Для правильного определения размера буфера следует воспользоваться функцией `GetEncoderParameterListSize`.

Для получения информации о поддерживаемых параметрах мы переработали программу по получению списков кодеров/декодеров, добавив в нее возможность вывода информации о категории, свойствах и возможных допустимых значениях параметров кодера записи. Данная программа кроме своего демонстрационного назначения будет полезна для быстрого освоения допустимых значений настроек конкретного установленного кодека.

Полный исходный код данной программы вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex11. Результат работы программы представлен на рис. 5.13.

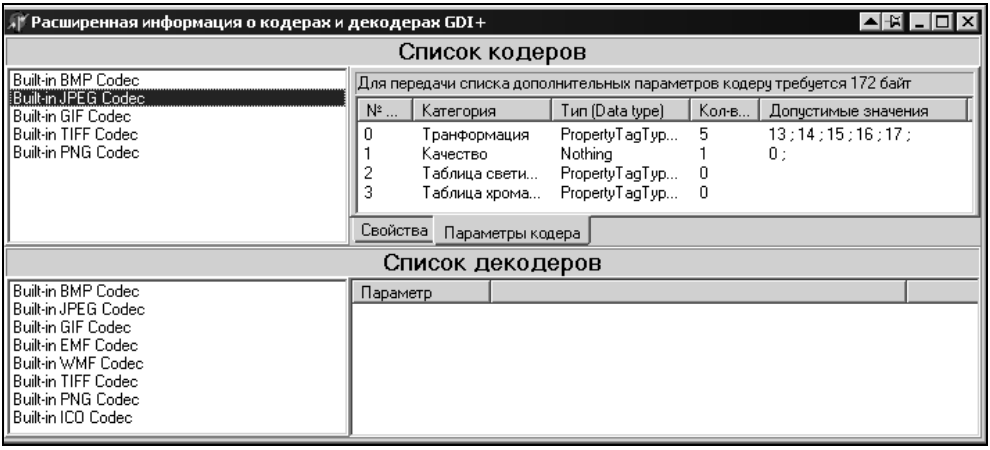


Рис. 5.13. Расширенная информация о кодерах и декодерах

Из результатов работы видно, что для кодера JPEG существует четыре дополнительных параметра: трансформация (EncoderTransformation), качество (EncoderQuality), таблица светимости (EncoderLuminanceTable), таблица хроматических данных (EncoderChrominanceTable). Так, параметр трансформации может принимать следующие допустимые значения: 13, 14, 15, 16, 17. Этим значениям соответствуют следующие элементы перечисления `EncoderValue`, описывающие допустимые геометрические преобразования при сохранении файлов формата JPEG:

- `EncoderValueTransformRotate90`
- `EncoderValueTransformRotate180`

- EncoderValueTransformRotate270
- EncoderValueTransformFlipHorizontal
- EncoderValueTransformFlipVertical

За правильность передачи аргументов дополнительных параметров отвечает перечисление EncoderValue. Варианты элементов EncoderValue представлены в табл. 5.5.

Таблица 5.5. *Элементы перечисления* `EncoderValue`

EncoderValueColorTypeCMYK	EncoderValueRenderNonProgressive
EncoderValueColorTypeYCCK	EncoderValueTransformRotate90
EncoderValueCompressionLZW	EncoderValueTransformRotate180
EncoderValueCompressionCCITT3	EncoderValueTransformRotate270
EncoderValueCompressionCCITT4	EncoderValueTransformFlipHorizontal
EncoderValueCompressionRle	EncoderValueTransformFlipVertical
EncoderValueCompressionNone	EncoderValueMultiFrame
EncoderValueScanMethodInterlaced	EncoderValueLastFrame
EncoderValueScanMethodNonInterlaced	EncoderValueFlush
EncoderValueVersionGif87	EncoderValueFrameDimensionTime
EncoderValueVersionGif89	EncoderValueFrameDimensionResolution
EncoderValueRenderProgressive	EncoderValueFrameDimensionPage

Более полную информацию по использованию дополнительных параметров кодеров можно получить в SDK.

Сохранение изображений

Для сохранения изображения в GDI+ предусмотрен метод Save класса Image, имеющий следующий вид:

```
function Save(filename: WideString; const clsidEncoder: TGUID;
   encoderParams: PEncoderParameters = nil): TStatus; overload;
```

В качестве параметров метод Save принимает Unicode-строку filename, указатель на GUID кодека clsidEncoder и необязательный параметр encoderParams, представляющий собой указатель на структуру EncoderParameters для настройки дополнительных параметров кодера.

Для демонстрации работы данного метода мы переработали программу по рисованию графических примитивов. В новой версии по нажатию кнопки производится рисование в растр, который в свою очередь сохраняется в JPEG-формате с помощью метода `Save`. Исходный код данного примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex12.

Конвертирование изображений в другой формат

Используя метод `Save` класса `Image`, программист может легко конвертировать изображение из одного формата в другой. Для демонстрации приводим небольшой пакетный конвертер из одного формата в другой. Конечно, данный конвертер никогда не заменит программы вроде ACDSee, но для учебных целей очень даже сгодится. Исходный код данного примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex13.

В листинге 5.11 сокращенно представлена процедура `Convert`, непосредственно выполняющая конвертирование изображения.

Листинг 5.11. Конвертирование изображения в другой формат

```
procedure TForm1.Convert(FileName, NewFileName, CoderMimeType: String);
var
   encoderClsid: TGUID;
   Image: TGPImage;
begin
   Image := TGPImage.Create(FileName);
   GetEncoderClsid(CoderMimeType, encoderClsid);
   image.Save(NewFileName, encoderClsid, nil);
   image.Free;
end;
```

Использование дополнительных параметров кодера JPEG

В настоящее время самым популярным форматом изображения стал формат JPEG. Массовое распространение данного формата в Интернете, в сфере цифровой фотографии стало возможным за счет малых размеров файла при сохранении довольно хорошего качества. Хотя JPEG и относится к классу форматов, сжимающих картинку с потерями, однако в любительской среде данный формат прочно занял одно из ведущих мест. Для программиста данный формат интересен еще и тем, что предоставляет множество дополнительных параметров, управляющих работой кодера при сохранении рисунка.

Настройка уровня компрессии

Одним из наиболее используемых параметров кодера JPEG является параметр, определяющий уровень сжатия (компрессии) исходного рисунка.

В листинге 5.12 приведен фрагмент программы, сжимающий оригинальный рисунок в формат JPEG с компрессией равной 50%. Полный исходный код примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex14.

Листинг 5.12. Настройка уровня компрессии JPEG

```
procedure TForm1.FormCreate(Sender: TObject);
var
  encoderClsid: TGUID;
  encoderParameters: TEncoderParameters;
  quality: ULONG;
  Image: TGPImage;
begin
  // Оригинальное изображение
  Image := TGPImage.Create('Образец.bmp');
  // Получаем CLSID кодера
  GetEncoderClsid('image/jpeg', encoderClsid);
  // Первоначально, перед сохранением мы должны установить
  // свойства объекта EncoderParameters.
  // Объект EncoderParameters отвечает за установку дополнит.
  // параметров кодера
  // и состоит из массива элементов EncoderParameter.
  // В нашем случае, нам нужен параметр, отвечающий за уровень
  // компрессии, этот параметр может принимать значения от 0 до 100.
  encoderParameters.Count := 1;
  encoderParameters.Parameter[0].Guid:= EncoderQuality;
  encoderParameters.Parameter[0].Type_ := EncoderParameterValueTypeLong;
  encoderParameters.Parameter[0].NumberOfValues := 1;
  // Устанавливаем сжатие JPEG с компрессией 50.
  quality := 50;
  encoderParameters.Parameter[0].Value := @quality;
  Image.Save('Out_050.jpg', encoderClsid, @encoderParameters);
  image.Free;
end;
```

Использование встроенных трансформаций

Другим интересным параметром кодера JPEG является параметр, позволяющий выполнять различные трансформации при записи, в частности повороты

на 90, 180 и 270°, а также отражения по горизонтали и вертикали. Интересным фактором работы кодера JPEG является следующая особенность: если загрузить файл JPEG с размерами, кратными 16, то при сохранении его с последующей трансформацией дополнительной потери качества не произойдет.

Демонстрационный пример использования встроенных трансформаций JPEG-кодера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex15.

Загрузка и сохранение многокадровых файлов

Как известно, некоторые графические форматы поддерживают хранение нескольких картинок в одном файле. Например, самым распространенным многокадровым форматом является формат GIF, применяющийся для анимации Web-страниц. В полиграфии распространен формат TIFF, который также может содержать в себе несколько изображений.

Ранее работать с многокадровыми форматами было довольно сложно: старый GDI практически не поддерживал работу с ними, и программистам приходилось прибегать к разбору данных форматов на низком уровне или использовать библиотеки сторонних разработчиков. В GDI+ работа с многокадровыми форматами ничем не отличается от обычной работы с рисунками. Более того, в GDI+ добавлены методы, позволяющие программисту производить различные манипуляции с фреймами.

> ***Примечание***
>
> Фрейм (кадр) — это отдельный рисунок, хранящийся в многокадровом изображении.

Перед началом работы с многокадровыми изображениями вы должны узнать число фреймов, находящихся в данном изображении. Сделать это можно с помощью метода `GetFrameCount` класса `Image`:

```
function GetFrameCount(const dimensionID: TGUID): UINT;
```

Данный метод в качестве параметра принимает GUID-константу, определяющую тип хранимых кадров. Возможные значения констант описаны в заголовочных файлах GDI+. Так, для файлов GIF необходима константа `FrameDimensionTime`, а для файлов TIFF — `FrameDimensionPage`. Для того чтобы вывести необходимый фрейм, нужно сначала его активировать с помощью метода `SelectActiveFrame` класса `Image`.

В примере, представленном в листинге 5.13, приведен исходный код программы, выводящий многокадровый рисунок по фреймам. Полный исходный код примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex16.

Листинг 5.13. Вывод многокадрового рисунка по фреймам

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  width           : UINT;        // Размер по ширине
  destinationRect : TGPRectF;    // Область
  I, Count        : Integer;
  FrameImg        : TGPImage;
begin
  // Создаем объект Graphics
  graphicsGDIPlus := TGPGraphics.Create(PaintBox1.Canvas.Handle);
  FrameImg := TGPImage.Create('DemoFrames.gif');
  // Определим размер по ширине
  width  := FrameImg.GetWidth;
  // Получаем кол-во фреймов
  Count := FrameImg.GetFrameCount(FrameDimensionTime);
  StaticText.Caption := 'Количество фреймов в изображении - ' +
                                                    IntToStr(Count);
  // Рисуем все кадры
  For I:= 0 to Count-1 do
  begin
      FrameImg.SelectActiveFrame(FrameDimensionTime, I);
      graphicsGDIPlus.DrawImage(FrameImg, (I * Width) + 1, 1);
  end;
  FrameImg.Free;
end;
```

Как вы видите, чтение кадров не представляет особой сложности в GDI+. Другое дело — сохранение множества кадров в многокадровый формат. К сожалению, в первой версии GDI+ многокадровое сохранение реализовано только для файлов формата TIFF. Создать анимированный GIF-файл средствами GDI+ пока не удается. Возможно, данная проблема будет решена в следующих версиях библиотеки.

Принцип сохранения многокадровых рисунков довольно прост: сначала нам нужно создать объект `Image`, который будет управлять процессом сохранения. Данный объект должен содержать первый кадр многокадрового изображения, после чего вы должны сохранить данный кадр с помощью метода `Save`. Для сохранения последующих кадров в том же файле нужно вызывать метод `SaveAdd`, передавая ему следующий кадр. Также можно добавлять кадры из других растров, вызывая одну из версий перегруженного метода `SaveAdd`, принимающего в качестве параметра указатель на объект класса `Image`. Для завершения процесса необходимо вызвать метод `SaveAdd` с особым параметром сохранения `EncoderValueFlush`.

В примере, представленном в листинге 5.14, мы создали TIFF-файл и сохранили в него три разных кадра. Исходный код данного примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex17.

Листинг 5.14. Запись многокадрового изображения

```
procedure TForm1.Button1Click(Sender: TObject);
var
  encoderParameters: TEncoderParameters;
  parameterValue   : TEncoderValue;
  MultiFrame, NewFrame  : TGPImage;
  encoderClsid     : TGUID;
  I                : Integer;
begin
   encoderParameters.Count := 1;
   // Инициализируем EncoderParameter.
   encoderParameters.Parameter[0].Guid := EncoderSaveFlag;
   encoderParameters.Parameter[0].Type_ := EncoderParameterValueTypeLong;
   encoderParameters.Parameter[0].NumberOfValues := 1;
   encoderParameters.Parameter[0].Value := @parameterValue;
   // Получаем CLSID кодера TIFF.
   GetEncoderClsid('image/tiff', encoderClsid);
   // Создаем 1 кадр.
   MultiFrame := TGPImage.Create('1.bmp');
   // Устанавливаем MultiFrmae
   parameterValue := EncoderValueMultiFrame;
   // Записываем начальный кадр
   MultiFrame.Save('DemoMultiFrame.tif', encoderClsid,
                                         @encoderParameters);
   // Записываем следующие кадры
   For I:= 2 to 3 do
   begin
       NewFrame := TGPImage.Create(IntToStr(I)+'.bmp');
       // Следующий кадр
       parameterValue := EncoderValueFrameDimensionPage;
       MultiFrame.SaveAdd(NewFrame, @encoderParameters);
       NewFrame.Free;
   end;

   // Закрываем многокадровый рисунок.
   parameterValue := EncoderValueFlush;
   MultiFrame.SaveAdd(@encoderParameters);
   // Не забудем высвободить память
   MultiFrame.Free;
end;
```

Работа с метаданными

Очередным новшеством в GDI+ стала возможность работать с метаданными, хранящимися в графических файлах. Метаданные представляют собой различную атрибутивную информацию, прикрепленную к данному изображению. Атрибутивная информация может содержать: авторские права, дату съемки или создания изображения. Для работы с метаданными библиотека GDI+ предоставляет класс `PropertyItem`, имеющий следующие свойства, представленные в табл. 5.6.

Таблица 5.6. *Свойства класса* `PropertyItem`

Свойство	Описание
`id`	Уникальный номер
`length`	Длина поля (в байтах)
`type`	Тип значения (один из типов `PropertyTagTypeXXXX`)
`value`	Указатель на данные

В настоящее время GDI+ поддерживает работу с метаданными только с форматами TIFF, JPEG, Exif и PNG.

Для получения информации о типах и форматах метаданных, хранимых в графических изображениях, мы разработали программу, представленную на рис. 5.14.

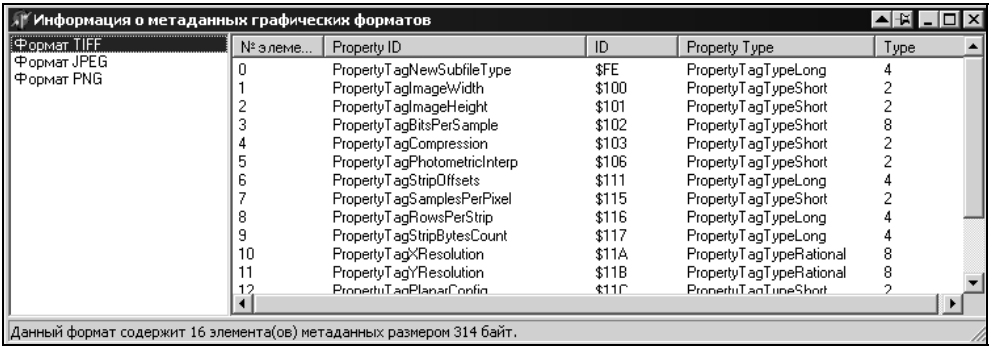


Рис. 5.14. Информация о метаданных

Полный исходный код данной программы можно найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex18.

Для чтения и записи метаданных из файла графического формата вы должны использовать методы `GetPropertyItem` и `SetPropertyItem` класса `Image`. В GDI+

жестко задан набор констант, которые идентифицируют свойства метаданных. Хотя некоторые константы и являются общими для всех поддерживаемых форматов, остальные, в основном, описывают метаданные только конкретного формата. Если вы попробуете задать в качестве аргумента константу, не поддерживаемую данным форматом, GDI+ попросту игнорирует ее. Более того, при использовании такой константы в методе `SetPropertyItem` класса `Image` данный метод вернет код возврата `PropertyNotSupported`, а при использовании в методе `GetPropertyItem` — `PropertyNotFound`.

В примере, представленном в листинге 5.15, демонстрируется чтение и сохранение заголовка (`ImageTitle`) изображения в метаданных. Исходный код данного примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex19.

Листинг 5.15. Работа с метаданными

```
procedure TForm1.ButtonSaveClick(Sender: TObject);
var
clsid: TGUID;
propertyValue: PChar;
Bitmap: TGPBitmap;
propertyItem: TPropertyItem;
begin
// Значение
propertyValue := PChar(Edit1.Text);
Bitmap:= TGPBitmap.Create('Demo.JPG');
GetEncoderClsid('image/jpeg', clsid);
propertyItem.id   := PropertyTagImageTitle; // Тип свойства
propertyItem.length := Length(propertyValue); // Длина значения
propertyItem.type_ := PropertyTagTypeASCII;  // Формат записи
propertyItem.value := propertyValue;     // Значение
bitmap.SetPropertyItem(propertyItem);
bitmap.Save('Demo_Metadata.JPG', clsid, nil);
Bitmap.Free;
end;

procedure TForm1.ButtonReadClick(Sender: TObject);
var
Image    : TGPImage;
propertyItem: PPropertyItem;
Size    : Integer;
begin
Image:= TGPImage.Create('Demo_Metadata.JPG');
```

```
// Получаем размер свойства метаданных PropertyTagImageTitle
size := image.GetPropertyItemSize(PropertyTagImageTitle);
IF Size = 0 then Exit;
// Выделяем память
GetMem(propertyItem ,Size);
// Считываем значение, хранимое в метаданных
image.GetPropertyItem(PropertyTagImageTitle, size, propertyItem);
IF propertyItem.type_ = PropertyTagTypeASCII then
begin
  Edit2.Text := PChar(propertyItem.value);
end;
FreeMem(propertyItem);
Image.Free;
end;
```

Как видно из приведенного примера, работа с метаданными не представляет особой сложности для программиста. Однако мы пока не рекомендуем использовать метаданные GDI+. Дело в том, что по нашим наблюдениям в GDI+ метаданные не совсем корректно работают. Например, после сохранения метаданных мы не смогли их прочитать ни в ACDSee, ни в Photoshop, и наоборот, при сохранении метаданных в ACDSee или Photoshop GDI+ наотрез отказался считывать их, что привело к появлению различных ошибок.

Использование Alpha-канала для создания эффектов прозрачности

Как было сказано в предыдущих разделах, в GDI+ появилась возможность работы с прозрачностью. Достигается это за счет введения Alpha-канала в палитру цветов GDI+. Для получения искомого цвета при использовании Alpha-каналов вы можете воспользоваться данной формулой:

```
displayColor = sourceColor × alpha / 255 + backgroundColor × (255 -
 alpha) / 255
```

Работа с Alpha-каналом при рисовании графических примитивов

Рисование прозрачных графических примитивов ничем не отличается от обычного рисования. Для установки прозрачности нужно лишь воспользоваться перегруженной функцией `MakeColor`, принимающей 4 параметра: Alpha-канал и три цветовые составляющие RGB при задании цвета в соответствующих конструкторах перьев и кистей. Демонстрационный пример вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex20.

Установка Alpha-канала для пикселов растра с помощью операции Recoloring

При работе с растрами в различных проектах иногда нам требовалось сделать их прозрачными. Ранее нам приходилось идти на различные ухищрения при программировании, однако с появлением GDI+ ситуация в корне изменилась. В новой библиотеке сделать это оказалось не сложно. Реализовать прозрачность растрового рисунка позволяет применение цветовых вычислений над каждым пикселом.

> ***Примечание***
>
> Цветовые вычисления, или иначе их еще называют `recoloring`, представляют собой операции матричной алгебры с цветовыми составляющими R, G, B. Alpha-канал в GDI+ равноправно с цветовыми составляющими участвует в данных вычислениях, что позволяет использовать `recoloring` для изменения прозрачности рисунка.

Выполнение цветовых вычислений в GDI+ осуществляется с помощью метода `SetColorMatrix` класса `ImageAttributes`. Сама матрица задается с помощью объекта `TColorMatrix`, являющегося матрицей, представленной в виде двухмерного массива размером 5×5 элементов (R, G, B, Alpha-канал и фиктивный элемент, необходимый для вычислений). Каждый элемент `TColorMatrix` является коэффициентом в диапазоне от 0 до 1.

При операциях с Alpha-каналом не следует забывать, что диапазон коэффициента от полной прозрачности (0) до полной непрозрачности (1). Так, для установки 20%-й прозрачности нужно использовать значение коэффициента 0,8, как в данной цветовой матрице:

$$\begin{bmatrix} 1,0 & 0,0 & 0,0 & 0,0 & 0,0 \\ 0,0 & 1,0 & 0,0 & 0,0 & 0,0 \\ 0,0 & 0,0 & 1,0 & 0,0 & 0,0 \\ 0,0 & 0,0 & 0,0 & 0,8 & 0,0 \\ 0,0 & 0,0 & 0,0 & 0,0 & 1,0 \end{bmatrix}$$

Число 0,8 в четвертой строке и четвертой колонке означает, что при умножении вектора из 5 элементов на эту матрицу четвертый элемент результирующего вектора будет равен исходному элементу, умноженному на 0,8. А т. к. данный элемент представляет собой значение Alpha-канала, то с помощью данной матрицы мы устанавливаем необходимую нам 20%-ю прозрачность.

Для демонстрации цветовых вычислений над Alpha-каналом нами предлагается демонстрационная программа, исходный код которой вы найдете на прилагаемом компакт-диске в каталоге Source\Ch05\Ex21.

Установка Alpha-канала прямой заменой цвета пикселов растра

Одним из интересных способов создания полупрозрачных растровых изображений является способ замены индивидуального цвета пиксела. Хотя данный способ и работает гораздо медленнее, чем способ установки Alpha-каналов с помощью матричной алгебры, зато он предоставляет огромные возможности по созданию различных графических эффектов.

Пример, представленный в листинге 5.16, демонстрирует технику замены значения Alpha-канала пикселов точечного рисунка. Для получения цвета каждого пиксела используется метод `GetPixel` класса `Image`, затем с помощью метода `SetValue` создается временный цвет, содержащий новое значение Alpha-канала. После этого метод `SetPixel` устанавливает новый цвет пиксела. Хочется также отметить, что оригинальный рисунок при этом не подвергается модификации, т. к. вся работа происходит в оперативной памяти.

Листинг 5.16. Прямая замена цвета

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  bitmap                    : TGPBitmap; // Битовое изображение
  ImageWidth, Imageheight   : UINT;      // Размер
  Row, Column               : Integer;
  color, colorTemp          : TGPColor;
  Pen                       : TGPPen;
  Pos,X,Y                       : Integer;
begin
  // Создаем объект Graphics
  graphicsGDIPlus := TGPGraphics.Create(PaintBox1.Canvas.Handle);
  Bitmap := TGPBitmap.Create('Истребитель.JPG');
  ImageWidth  := bitmap.GetWidth;
  ImageHeight := bitmap.GetHeight;
  // Рисуем сетку
  Pen := TGPPen.Create(MakeColor(0,0,0));
  Pos := 0;
  While Pos <= ImageWidth do
  begin
      graphicsGDIPlus.DrawLine(Pen,Pos,0,Pos,ImageHeight - 1);
      Inc(Pos,15);
  end;
  Pos := 0;
  While Pos <= ImageHeight do
  begin
```

```
      graphicsGDIPlus.DrawLine(Pen,0,Pos,ImageWidth - 1,Pos);
      Inc(Pos,15);
  end;

  // Проходим по строкам растрового изображения
  for Row := ImageHeight - 1 downto 0 do
  begin
    // Проходим по колонкам растрового изображения до
    // половины изображения по горизонтали
    for Column := 0 to ImageWidth div 2 do
    begin
      // Получаем цвет
      bitmap.GetPixel(Column, Row, color);
      // Создаем временный цвет
      colorTemp := MakeColor((255 * Column div ImageWidth) * 2 ,
                   GetRed(color),GetGreen(color),GetBlue(color));
      // Устанавливаем новый цвета пиксела
      bitmap.SetPixel(Column, Row, colorTemp);
    end;
  end;
  // Рисуем модифицированное изображение
  graphicsGDIPlus.DrawImage(bitmap, 30, 0, ImageWidth, ImageHeight);
  // Очищаем память
  Pen.Free;
  Bitmap.Free;
  graphicsGDIPlus.Free;
end;
```

Данный пример создает плавный переход от полной прозрачности до полной непрозрачности изображения. Переход начинается с левого края и заканчивается серединой изображения. Полный исходный код вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex22.

Работа с текстом

Для работы с текстом библиотека GDI+ представляет несколько классов, которые формируют основу для вывода текста. Основную роль по выводу текста берет на себя класс `Graphics`, реализующий три перегруженных версии метода `DrawString`. Различные версии данного метода позволяют программисту задавать форматирование выводимого текста, а также определять область вывода.

Вспомогательную роль играют классы, задающие различные параметры. Так, для установки шрифтов задействуются классы: `FontFamily`, `Font`,

InstalledFontCollection и PrivateFontCollection, а для управления форматированием текста используется класс StringFormat.

Установка шрифта для вывода текста

Прежде чем вывести текст, вы должны создать экземпляры следующих классов: FontFamily и Font. Класс FontFamily отвечает за выбор шрифта из семейства шрифтов, а класс Font определяет характеристики шрифта, такие как размер или стиль шрифта.

Так, например, следующий код создает шрифт "Times New Roman" размером 32 пиксела:

```
FontFamily := TGPFontFamily.Create('Times New Roman');
Font := TGPFont.Create(FontFamily, 32, FontStyleRegular, UnitPixel);
```

При создании шрифта в конструкторе Font в качестве первого параметра задается указатель на созданный объект FontFamily. Второй параметр определяет размер шрифта, заданный в единицах, которые в свою очередь задаются в четвертом параметре. Третий параметр устанавливает стиль шрифта.

При установке стиля шрифта используются значения, заданные в перечислении FontStyle и представленные в табл. 5.7.

Таблица 5.7. *Значения* FontStyle

Значение	Описание
FontStyleRegular	Обычное начертание
FontStyleBold	Жирный
FontStyleItalic	Курсив
FontStyleBoldItalic	Жирный курсив
FontStyleUnderline	Подчеркнутый
FontStyleStrikeout	Зачеркнутый

При установке единиц размера шрифта используются значения, заданные в перечислении Unit и представленные в табл. 5.8.

Таблица 5.8. *Значения* Unit

Значение	Описание
UnitWorld	Устанавливает размер, не привязанный к конкретному устройству вывода

Таблица 5.8 *(окончание)*

Значение	Описание
UnitDisplay	Задает единицу размера, связанную с устройством вывода. Например, если устройство вывода — монитор, то единица размера равна одному пикселу (UnitPixel)
UnitPixel	Устанавливает размер в пикселах
UnitPoint	Устанавливает размер, равный 1/72 дюйма
UnitInch	Устанавливает размер в дюймах
UnitDocument	Устанавливает размер, равный 1/300 дюйма
UnitMillimeter	Устанавливает размер в миллиметрах

Вывод текста

Для вывода текста в классе Graphics реализовано три перегруженных версии метода DrawString. Рассмотрим их подробнее. Для простого вывода текста используется следующая реализация версии метода:

```
function DrawString(string_: WideString; length: Integer;
                    font: TGPFont;  const origin: TGPPointF;
                    brush: TGPBrush): TStatus; overload;
```

Для вывода текста с последующим форматированием строки GDI+ предоставляет следующие реализации:

```
function DrawString(string_: WideString; length: Integer;
                    font: TGPFont; const layoutRect: TGPRectF;
                    stringFormat: TGPStringFormat; brush: TGPBrush):
                        TStatus; overload;
function DrawString(string_: WideString; length: Integer;
                    font: TGPFont; const origin: TGPPointF;
                    stringFormat: TGPStringFormat; brush: TGPBrush):
                                          TStatus; overload;
```

Метод DrawString принимает в качестве параметров вызова следующие аргументы, представленные в табл. 5.9.

В листинге 5.17 вашему вниманию представлена часть исходного кода примера, производящего простой вывод текста без форматирования. Полный исходный код данного примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex23.

Таблица 5.9. *Параметры метода* `DrawString`

Параметр	Описание
`String_`	Указатель на строку текста, предназначенную для вывода
`Length`	Длина строки текста. Если параметр имеет значение равное –1, то длина строки рассчитывается автоматически по нуль-терминатору в конце строки
`font`	Указатель на класс `Font`, устанавливающий шрифт для выводимого текста
`Origin`	Параметр задает координаты начала вывода строки
`LayoutRect`	Параметр задает область вывода строки
`StringFormat`	Указатель на класс `StringFormat`, устанавливающий последующее форматирование при выводе текста. Если данный параметр имеет значение `Nil`, то форматирование текста не происходит
`Brush`	Указатель на класс кисти, обеспечивающий необходимую заливку букв выводимого текста

Листинг 5.17. Простой вывод текста

```
 // Выбираем шрифт из семейства шрифтов
FontFamily:= TGPFontFamily.Create('Times New Roman');
// Создаем шрифт высотой 24 пиксела
Font := TGPFont.Create(fontFamily, 24, FontStyleRegular,
          UnitPixel);
// Перо
SolidBrush:= TGPSolidBrush.Create(MakeColor(255, 0, 0, 255));
// Вывод
PointF := MakePoint(5.0, 10.0); // Начало вывода текста
GraphicsGDIPlus.DrawString('Вывод текста', -1, font,
             pointF, solidBrush);
// Выводим в определенную область
Str := 'Вывод текста в область прямоугольника.';
RectF:= MakeRect(160.0, 10.0, 120.0, 140.0);
GraphicsGDIPlus.DrawString(str, -1, font, rectF, nil, solidBrush);
```

Форматирование при выводе текста

Очень часто при выводе текста требуется задать ему необходимое форматирование, например, установить выравнивание текста по центру. До появления GDI+ разработчикам приходилось писать собственные функции и методы, производящие необходимое форматирование при выводе. В GDI+ для реше-

ния этой проблемы был выделен целый класс `StringFormat`, заключающий в себе довольно большое количество методов и свойств, позволяющих управлять форматированием текста. Подробное описание всех методов и свойств класса `StringFormat` вы можете найти в SDK. Здесь же мы остановимся на самых основных.

Выравнивание текста

Выравнивание текста в области вывода задается двумя методами `SetAlignment` и `SetLineAlignment`. Первый метод задает выравнивание текста по горизонтали, второй — по вертикали. Оба этих метода в качестве параметра, задающего форматирование, принимают следующие значения перечисления `StringAlignment`, представленные в табл. 5.10.

Таблица 5.10. *Значения* `StringAlignment`

Значение	Для `SetAlignment`	Для `SetLineAligment`
`StringAlignmentNear`	По левому краю	По верху
`StringAlignmentCenter`	По ширине	По ширине
`StringAlignmentFar`	По правому краю	По низу

Для демонстрации работы этих методов мы разработали программу, результат работы которой вы видите на рис. 5.15.

Рис. 5.15. Форматирование текста

Полный исходный код примера, производящего форматирование, вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex24.

Использование таб-стопов (табуляторов)

Одной из интересных особенностей класса `StringFormat` является поддержка разбиения текста на колонки с помощью таб-стопов (табуляторов).

Примечание

Таб-стоп (табулятор) — это символ табуляции, имеющий код #9 в таблице символов ASCII.

Используя табуляторы в выводимых строках, вы легко можете задать вывод текста в виде колонок и таблиц. Для этого вам нужно воспользоваться методом `SetTabStops` класса `StringFormat`:

```
function SetTabStops(firstTabOffset: Single; count: Integer;
          tabStops: PSingle): TStatus;
```

Данный метод устанавливает ширину колонок текста, разделенного символами табуляции. В качестве аргументов метод принимает следующие параметры, представленные в табл. 5.11.

Таблица 5.11. *Параметры* `SetTabStops`

Параметр	Описание
`firstTabOffset`	Начальное положение. Если данное значение равно нулю, то начальное положение устанавливается равным левой границе области вывода
`count`	Количество элементов в массиве
`tabStops`	Указатель на массив, хранящий значения ширины колонок табуляторов

Пример использования метода `SetTabStops` для установки ширины колонок представлен в листинге 5.18.

Листинг 5.18. Использование табуляторов при выводе текста

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  StringFormat: TGPStringFormat;
  // Массив для хранения ширины колонок таб-стопов
  Tabs        : packed array [0..3] of Single;
begin
  // Форматирование
  StringFormat:= TGPStringFormat.Create();
  // Установка ширины колонок таб-стопов
  Tabs[0] := 150;  Tabs[1] := 60;
  Tabs[2] := 100;  Tabs[3] := 80;
  StringFormat.SetTabStops(0,3,@tabs);
end;
```

Рис. 5.16. Вывод текста в колонки

На рис 5.16 представлен результат работы демонстрационной программы по выводу текста, разделенного табуляторами в виде таблицы. Полный исходный код этого примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex25.

Дополнительные настройки форматирования

Для установки дополнительных настроек форматирования выводимого текста в классе `StringFormat` предусмотрен метод `SetFormatFlags`. Данный метод принимает различные значения перечисления `StringFormatFlags`. Можно также отметить, что значения флагов `StringFormatFlags` в некоторых случаях могут быть объединены вместе. Описание наиболее интересных значений данного перечисления приведено в табл. 5.12. Более подробную информацию вы можете получить в SDK.

Таблица 5.12. *Значения* `StringFormatFlags`

Значение	Описание
`StringFormatFlagsDirectionRightToLeft`	При установке данного флага GDI+ осуществляет вывод текста справа налево, что характерно для арабских языков. По умолчанию вывод осуществляется слева направо
`StringFormatFlagsDirectionVertical`	Установка этого флага влечет за собой вывод текста по вертикали. По умолчанию вывод текста осуществляется по горизонтали
`StringFormatFlagsMeasureTrailingSpaces`	Данный флаг влияет на работу метода `MeasureString` класса `Graphics`, представляющего собой расширенный аналог функций `TextHeight` и `TextWidth` старого GDI. Метод позволяет получить информацию о количестве символов и строк текста, которые свободно могут разместиться в области вывода. По умолчанию количество знаков в строке, которая не полностью отрезается границей области вывода, не

Таблица 5.12 *(окончание)*

Значение	Описание
	суммируется в методе `MeasureString`. Установка данного флага включает данное суммирование. Более подробно о работе метода `MeasureString` будет рассказано далее
`StringFormatFlagsNoWrap`	Установка данного флага запрещает перенос текста на следующую строку
`StringFormatFlagsLineLimit`	Установка данного флага запрещает вывод строки, которая не полностью попадает в область вывода и обрезается ее границей
`StringFormatFlagsNoClip`	Установка данного флага разрешает полный вывод строки, которая не полностью попадает в область вывода и должна обрезаться. Вывод части строки происходит уже за пределами области вывода

Для демонстрации работы флагов `StringFormatFlags` мы написали небольшое тестовое приложение, позволяющее вам наглядно увидеть влияние, производимое установкой этих флагов на вывод текста. Результат работы данного теста вы можете увидеть на рис. 5.17.

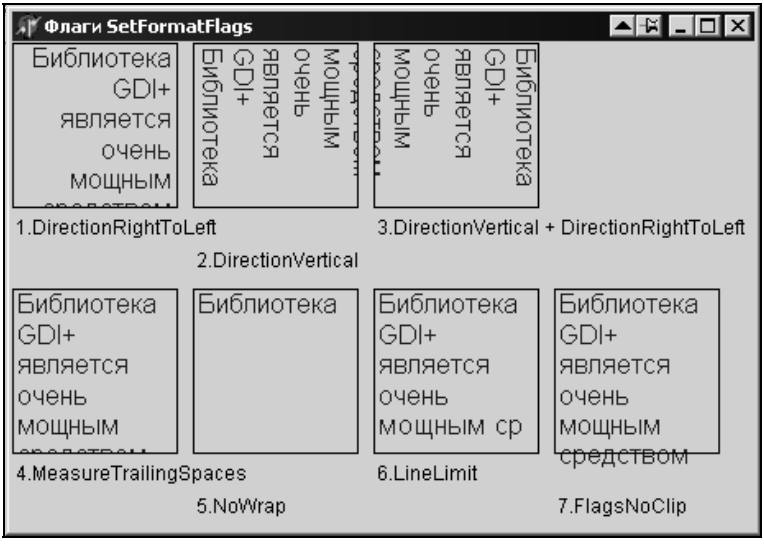


Рис. 5.17. Влияние флагов `StringFormatFlags` на вывод текста

Полный исходный код примера вы найдете на прилагаемом компакт-диске в каталоге Source\Ch05\Ex26.

Получение информации о метрике шрифта

Очень часто при выводе текста требуется получить различную информацию о метрике шрифта. Для этого в классе `FontFamily` предусмотрены следующие методы:

- `GetEmHeight` — размер шрифта;
- `GetCellAscent` — высота подъема;
- `GetCellDescent` — высота спуска;
- `GetLineSpacing` — высота межстрочного интервала.

Данные методы возвращают информацию в единицах дизайна (design unit), привязанных к устройству вывода.

На рис. 5.18 показана схема, объясняющая работу данных методов.

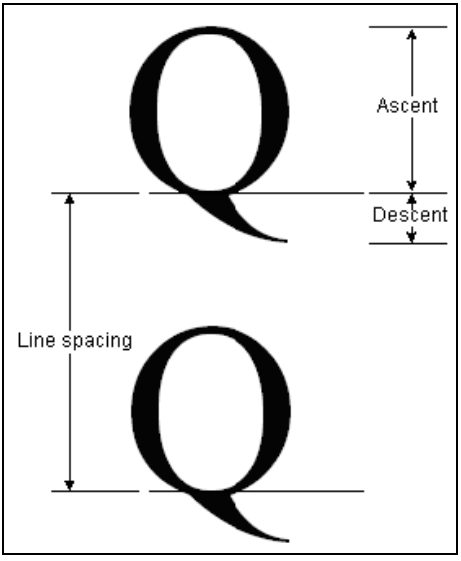


Рис. 5.18. Метрики шрифта

Для демонстрации работы этих методов мы разработали небольшую программу, позволяющую получать метрики различных шрифтов, установленных в системе. Исходный код примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex27.

В листинге 5.19 представлен сокращенный код данного примера.

Листинг 5.19. Получение метрик шрифтов

```
InfoString := format('Font.GetSize = %f.', [font.GetSize]);

InfoString := format('FontFamily.GetEmHeight = %d. единиц ' +
            'дизайна',[fontFamily.GetEmHeight(font.GetStyle)]);

ascent := fontFamily.GetCellAscent(font.GetStyle);
ascentPixel := font.GetSize * ascent /
               fontFamily.GetEmHeight(font.GetStyle);
InfoString := format('FontFamily.GetCellAscent (подъем) = %d ' +
              'единиц дизайна (%f пикселов).', [ascent, ascentPixel]);

descent := fontFamily.GetCellDescent(font.GetStyle);
descentPixel := font.GetSize * descent /
                fontFamily.GetEmHeight(font.GetStyle);
InfoString := format('FontFamily.GetCellDescent (спуск) = %d единиц' +
              'дизайна (%f пикселов).', [descent, descentPixel]);

lineSpacing := fontFamily.GetLineSpacing(font.GetStyle);
lineSpacingPixel := font.GetSize * lineSpacing /
                fontFamily.GetEmHeight(font.GetStyle);
InfoString := format('FontFamily.GetLineSpacing (межстрочный '+
              'интервал) = %d единиц дизайна (%f '+
              'пикселов).',[lineSpacing, lineSpacingPixel]);
```

Обратите внимание на то, что в данном примере производится перерасчет единиц дизайна к пикселам экрана.

Сглаживание шрифтов при выводе

Помимо традиционного способа вывода шрифта, библиотека GDI+ предоставляет в пользование более качественные способы вывода шрифта, основанные на новой технологии сглаживания шрифтов Microsoft ClearType. Данная технология доступна только в Microsoft Windows XP и Microsoft .NET. Технология ClearType обеспечивает более качественное сглаживание шрифта, улучшает удобочитаемость текста на современных цифровых мониторах и портативных компьютерах.

Для установки необходимого режима сглаживания шрифтов нужно воспользоваться методом `SetTextRenderingHint` класса `Graphics`. В качестве параметра данный метод принимает одно из значений перечисления `TextRenderHint`, представленных в табл. 5.13.

Таблица 5.13. Значения `TextRenderingHint`

TextRenderingHintSystemDefault	TextRenderingHintAntiAliasGridFit
TextRenderingHintSingleBitPerPixelGridFit	TextRenderingHintAntiAlias
TextRenderingHintSingleBitPerPixel	TextRenderingHintClearTypeGridFit

Наиболее качественную прорисовку шрифта обеспечивают значения: `TextRenderingHintAntiAlias` и `TextRenderingHintClearTypeGridFit`. Установка `TextRenderingHintClearTypeGridFit` обеспечивает наилучшее сглаживание для большинства мониторов при выводе небольших по высоте строк текста. Параметр `TextRenderingHintAntiAlias` обеспечивает лучшее качество вывода для вращаемого текста.

В листинге 5.20 приведен пример установки различных режимов сглаживания. Результат сглаживания вы можете увидеть на рис. 5.19.

Листинг 5.20. Различные режимы сглаживания

```
var
  graphicsGDIPlus : TGPGraphics;
  FontFamily: TGPFontFamily;
  Font: TGPFont;
  SolidBrush: TGPSolidBrush;
begin
  graphicsGDIPlus   := TGPGraphics.Create(DC);
  FontFamily := TGPFontFamily.Create('Times New Roman');
  Font       := TGPFont.Create(FontFamily, 32, FontStyleRegular,
                                               UnitPixel);
  SolidBrush := TGPSolidBrush.Create(MakeColor(255, 0, 0, 255));
  graphicsGDIPlus.SetTextRenderingHint
                                  (TextRenderingHintSingleBitPerPixel);
  graphicsGDIPlus.DrawString('SingleBitPerPixel', -1, font,
                             MakePoint(10.0, 10.0), solidBrush);

  graphicsGDIPlus.SetTextRenderingHint(TextRenderingHintAntiAlias);
  graphicsGDIPlus.DrawString('AntiAlias', -1, font,
                             MakePoint(10.0, 60.0), solidBrush);
  graphics.Free;
  FontFamily.Free;
  Font.Free;
  SolidBrush.Free;
```

Рис. 5.19. Результат сглаживания шрифта

Использование метода *MeasureString*

В данном разделе нам хотелось бы подробно остановиться на методе `MeasureString` класса `Graphics`. Дело в том, что данный метод является очень полезным для программиста, работающего с выводом текста, т. к. позволяет получить информацию о количестве символов и строк текста, которые свободно могут разместиться в области вывода. Поясним сказанное на примере. Положим, вы разрабатываете приложение, которое должно выводить на экран большой объем текстовых данных и вам нужно обеспечить возможность навигации, как в текстовых редакторах.

При выводе части текста с помощью метода `DrawString` перед вами вскоре станет задача узнать, сколько символов и строк текста уместилось в данной области вывода при использовании выбранного шрифта. Решить данную задачу вам поможет метод `MeasureString`, имеющий четыре перегруженных версии вызовов.

В данном разделе мы не будем подробно останавливаться на всех версиях этого метода, а выберем для примера и разберем более подробно следующий:

```
function MeasureString(string_: WideString;
                       length: Integer;
                       font: TGPFont;
                       const layoutRect: TGPRectF;
                       stringFormat: TGPStringFormat;
                       out boundingBox: TGPRectF;
                       codepointsFitted: PInteger = nil;
                       linesFilled: PInteger = nil): TStatus; overload;
```

В качестве параметров метод принимает аргументы, представленные в табл. 5.14.

Таблица 5.14. *Параметры метода* `MeasureString`

Параметр	Описание
`string_`	Указатель на строку текста, предназначенную для вывода
`length`	Длина строки текста

Таблица 5.14 *(окончание)*

Параметр	Описание
`font`	Указатель на класс `Font`, устанавливающий шрифт для выводимого текста
`layoutRect`	Область вывода строки
`stringFormat`	Указатель на класс `StringFormat`, устанавливающий форматирование при выводе текста
`boundingBox`	Параметр возвращает реальную область вывода данного текста
`codepointsFitted`	Ссылка на переменную типа `Integer`, куда будет возвращено количество символов вместе с пробелами, входящими в заданную область `LayoutRect`
`linesFilled`	Ссылка на переменную типа `Integer`, куда будет возвращено количество строк, входящих в заданную область `LayoutRect`

Для демонстрации работы данного метода мы разработали тестовое приложение, результат работы которого представлен на рис. 5.20.

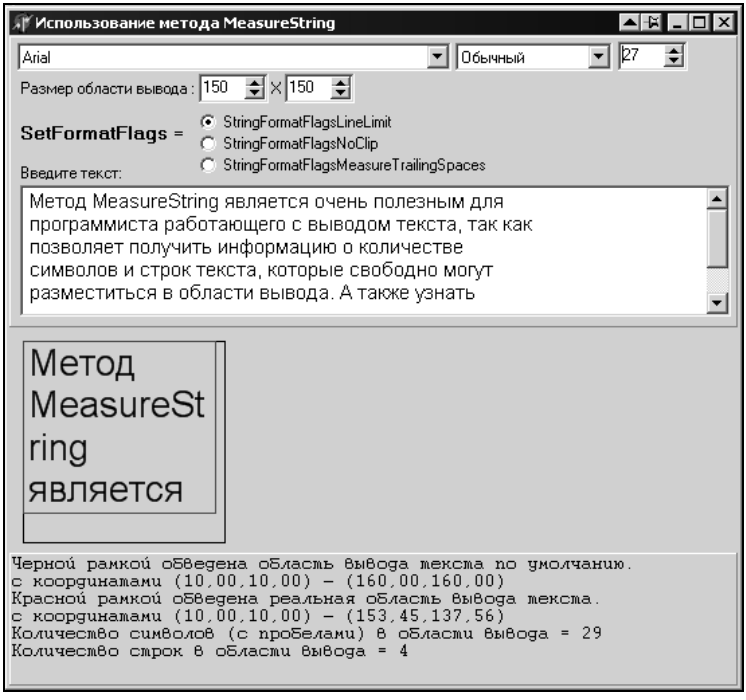


Рис. 5.20. Тестирование метода `MeasureString`

Полный исходный код примера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex28.

Координатная система

Библиотека GDI+ использует три координатных пространства: мировое, страничное и координатное пространство устройства вывода (ПУВ).

Так, когда вы вызываете метод `DrawLine` для рисования линии, в качестве параметров координат вы передаете значения мировых координат. Прежде чем GDI+ произведет рисование линии на экране, переданные координаты пройдут через ряд преобразований. Сначала мировые координаты будут преобразованы к координатам страницы, после чего произойдет преобразование к координатам устройства. Данная технология работы позволяет производить различные координатные преобразования.

Поясним сказанное на примере. Допустим, что вы хотите работать с локальной системой координат, начало которой находится в точке (100, 50). По умолчанию точка отсчета координат экрана ведется от верхнего левого угла. После установки новой точки отсчета мировых координат с помощью метода трансформации `TranslateTransform` класса `Grapchics` последующий вызов метода `DrawLine` с координатами (0, 0) — (160, 80) совершит ряд преобразований, в результате чего координаты примут значения, представленные в табл. 5.15 и на рис. 5.21.

Таблица 5.15. *Значения координат в координатных пространствах*

Координатное пространство	Значения координат
Мировое	0, 0 — 160, 80
Страничное	100, 50 — 260, 130
ПУВ	100, 50 — 260, 130

Помимо установок собственных локальных систем координат, GDI+ также позволяет изменять единицы измерения страничной системы координат и масштаб координатной сетки. Для этого в классе `Graphics` предусмотрены следующие методы, показанные в табл. 5.16.

Таблица 5.16. *Методы класса `Graphics` для работы с единицами измерений*

Метод	Описание
`SetPageUnit`	Метод устанавливает единицу измерения страничной системы координат. В качестве параметра метод принимает значение перечисления `Unit`

Таблица 5.16 *(окончание)*

Метод	Описание
`GetPageUnit`	Данный метод позволяет получить информацию о текущей единице измерения
`SetPageScale`	Данный метод устанавливает коэффициент масштабирования значений координат для страничных трансформаций
`GetPageScale`	Метод позволяет получить информацию о текущем коэффициенте масштабирования
`GetDpiX`	Метод получает горизонтальную разрешающую способность устройства отображения в точках на дюйм
`GetDpiY`	Метод получает вертикальную разрешающую способность устройства отображения в точках на дюйм

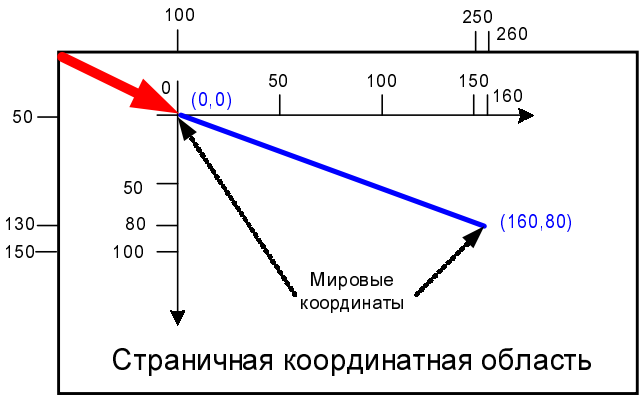


Рис. 5.21. Представление линии в координатных областях

Использование метода `SetPageUnit` предоставляет программисту возможность применять естественные единицы измерения длины.

Допустим, что некое устройство отображения имеет следующие характеристики разрешающей способности: 96 точек/дюйм по горизонтали и 96 точек/дюйм по вертикали. Так, используя метод `SetPageUnit` с параметром установки единиц измерения в дюймах, мы получим, что при рисовании линии, например, с координатами (0, 0) — (2, 1) координаты концов линии примут следующие значения координатных пространств, представленные в табл. 5.17.

Таблица 5.17. *Значения координат в координатных пространствах*

Координатное пространство	Значения координат
Мировое	0, 0 — 2, 1
Страничное	0, 0 — 2, 1
ПУВ	0, 0 — 192, 96

Употребление же метода `SetPageScale` дает широкие возможности масштабирования при выводе различных объектов GDI+, а также установки собственных единиц измерения. Так, в перечислении `Unit` нет значения, устанавливающего единицы измерения в сантиметрах, зато есть значение `UnitMillimeter`, задающее измерение в миллиметрах. Комбинируя метод `SetPageUnit` с параметром `UnitMillimeter` и `SetPageScale` с коэффициентом 10, вы фактически устанавливаете собственные единицы измерения — сантиметры, а используя коэффициент 100 — дециметры.

Путем комбинирования мировых и страничных преобразований (трансформаций) можно добиться различных графических эффектов. Например, мы хотим использовать дюймы в качестве единицы измерения и хотим сместить начало координат на 2 дюйма вправо и 0,5 дюйма вниз относительно начала координат клиентского поля.

Ниже приведен пример вывода на экран линии, идущей от координаты (0, 0) к (2, 1) в такой системе.

```
myGraphics.SetPageUnit(UnitInch);
myGraphics.TranslateTransform(2.0, 0.5);
myGraphics.DrawLine(myPen, 0, 0, 2, 1);
```

Координаты концов линии примут следующие значения координатных пространств, представленные в табл. 5.18 и на рис. 5.22.

Таблица 5.18. *Значения координат в координатных пространствах*

Координатное пространство	Значения координат
Мировое	0, 0 — 2, 1
Страничное	2, 0,5 — 4, 1,5
ПУВ	192, 48 — 384, 144

Для конвертирования координат из одного координатного пространства в другое в GDI+ существует метод `TransformPoints` класса `Graphics`. Данный

метод в качестве параметров `DestSpace` и `SrcSpace` принимает одно из значений перечисления `CoordinateSpace`. Параметр `DestSpace` определяет пространство-получатель, а параметр `SrcSpace` — пространство-источник. Координаты точек, подлежащие пересчету, задаются в параметре `Pts`, представляющем собой ссылку на массив точек. Параметр `Count` в свою очередь задает количество элементов в массиве точек.

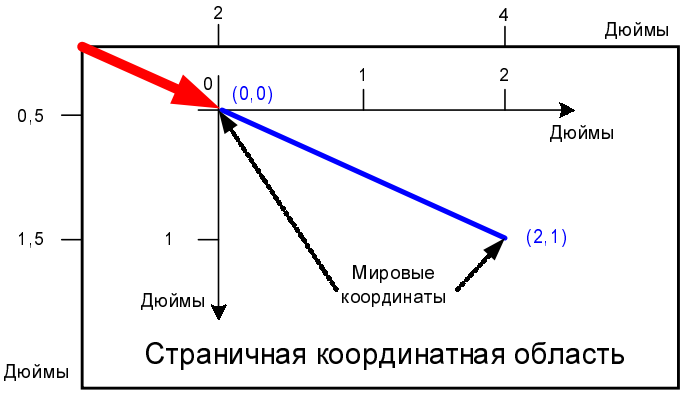


Рис. 5.22. Представление линии в координатных областях

Пример использования метода `TransformPoints` вы можете увидеть в листинге 5.21. В данном примере на основе координаты точки, расположенной в пространстве устройства отображения, с помощью метода `TransformPoints` мы получаем координату в координатном пространстве страницы.

Листинг 5.21. Перевод координат

```
var
  GraphicsGDIPlus : TGPGraphics;
  PTS             : PGPPoint;
Begin
  GraphicsGDIPlus := TGPGraphics.Create(Form1.Handle);
  GraphicsGDIPlus.SetPageUnit(UnitPixel);
  GraphicsGDIPlus.SetPageScale(10); // Коэффициент масштабирования - 10
  New(PTS);
  PTS^.X  := 100; // Точка в координатном пространстве устройства
  PTS^.Y  := 100; // отображения

  // Не перепутайте - 1 параметр является получателем, 2 - источником
  GraphicsGDIPlus.TransformPoints(CoordinateSpacePage,
                                  CoordinateSpaceDevice,PTS,1);
```

```
  // Получаем координаты точки в координатном пространстве страницы
  ShowMessage(FloatToStr(PTS^.X) + '. ' + FloatToStr(PTS^.Y));
                // Возвращаемое значение будет равно X=10, Y:=10

  Dispose(PTS);
  GraphicsGDIPlus.Free;
end;
```

Преобразования (трансформации) объектов

Библиотека GDI+ в своей архитектуре поддерживает следующие аффинные преобразования объектов GDI+: масштабирование, вращение (поворот), отражение, трансляцию и сдвиг. Прежде чем мы подробно остановимся на методах GDI+, немного углубимся в теорию матричных преобразований, т. к. все преобразования графических объектов происходят в GDI+, реализуются именно с помощью матриц.

Матричное представление преобразований

Матрицей $m \times n$ называется прямоугольная таблица чисел, содержащая m строк и n столбцов. Числа в матрице называются ее *элементами*. Далее приведены примеры различных матриц:

$$\underset{2\times2}{\begin{pmatrix}3 & 1\\ 2 & 5\end{pmatrix}};\ \underset{2\times3}{\begin{pmatrix}6 & 1 & 4\\ 2 & 7 & 0\end{pmatrix}};\ \underset{1\times2}{\begin{pmatrix}3 & 7\end{pmatrix}};\ \underset{3\times1}{\begin{pmatrix}3\\ 4\\ 1\end{pmatrix}};\ \underset{3\times3}{\begin{pmatrix}3 & 2 & 1\\ 2 & 3 & 2\\ 4 & 1 & 5\end{pmatrix}}.$$

Две матрицы одинакового размера можно складывать путем сложения соответствующих элементов:

$$\begin{pmatrix}2 & 5\end{pmatrix}+\begin{pmatrix}3 & 1\end{pmatrix}=\begin{pmatrix}5 & 6\end{pmatrix},$$

$$\begin{pmatrix}1 & 3\\ 0 & 2\end{pmatrix}+\begin{pmatrix}1 & 2\\ 3 & 1\end{pmatrix}=\begin{pmatrix}2 & 5\\ 3 & 3\end{pmatrix}.$$

Матрица $m \times n$ может быть умножена на матрицу $n \times p$, результатом будет являться матрица $m \times p$, т. е. число строк результирующей матрицы должно быть равно числу строк первой матрицы, а число колонок результирующей матрицы — числу колонок второй. Например, результатом умножения матрицы 4×2 на матрицу 2×3 будет матрица 4×3.

Координаты точек и векторов на плоскости (и в пространстве) могут быть записаны в виде матрицы 1×2 или 2×1 (1×3 или 3×1 соответственно). Например, точка с координатой (2, 5) может быть записана в виде матрицы $\begin{pmatrix} 2 \\ 5 \end{pmatrix}$.

Скалярное произведение векторов определяется следующим образом:

$$(a, b) \bullet (c, d) = ac + bd$$
$$(a, b, c) \bullet (d, e, f) = ad + be + cf$$

Например, скалярное произведение векторов (2, 3) и (5, 4) равно (2)(5) + (3)(4) = 22, а скалярное произведение векторов (2, 5, 1) и (4, 3, 1) равно (2)(4) + (5)(3) + (1)(1) = 24.

Заметим, что скалярным произведением векторов является число, а не вектор.

Следует также заметить, что в операциях умножения должны участвовать векторы одинаковой размерности.

Записью $A(i, j)$ обозначают элемент матрицы A, находящийся на пересечении строки i и столбца j.

Например, $A(3, 2)$ элемент матрицы A, находящийся в 3-й строке и во 2-м столбце. Произведением двух матриц A и B называется матрица C, элементы которой вычисляются по следующему правилу:

$$C(i, j) = (\text{строка } i \text{ матрицы } A) \bullet (\text{столбец } j \text{ матрицы } B).$$

Далее вы можете увидеть примеры произведений двух матриц:

$$\begin{pmatrix} 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 2 & 4 \\ 5 & 1 \end{pmatrix} = \begin{pmatrix} 17 & 14 \end{pmatrix};$$

$$\begin{pmatrix} 2 & 3 \end{pmatrix} \begin{pmatrix} 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 16 \end{pmatrix};$$

$$\begin{pmatrix} 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 2 & 4 \\ 5 & 1 \end{pmatrix} = \begin{pmatrix} 17 & 14 \end{pmatrix}.$$

Если представить точку на плоскости в виде матрицы 1×2, то можно легко трансформировать ее путем умножения на матрицу 2×2.

Следующие иллюстрации показывают некоторые преобразования точки (2, 1):

- Масштабирование — растяжение вдоль оси X (рис. 5.23).

$$\begin{pmatrix}2 & 1\end{pmatrix}\begin{pmatrix}3 & 0\\ 0 & 1\end{pmatrix}=\begin{pmatrix}6 & 1\end{pmatrix}$$

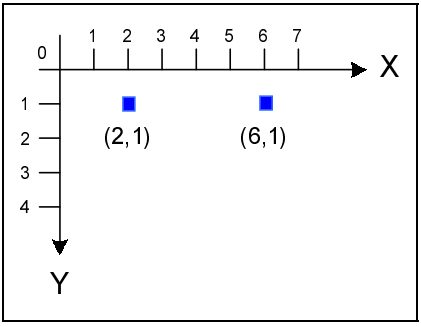


Рис. 5.23. Масштабирование

- Поворот на 90° в отрицательном направлении (рис. 5.24).

$$\begin{pmatrix}2 & 1\end{pmatrix}\begin{pmatrix}0 & 1\\ -1 & 0\end{pmatrix}=\begin{pmatrix}-1 & 2\end{pmatrix}$$

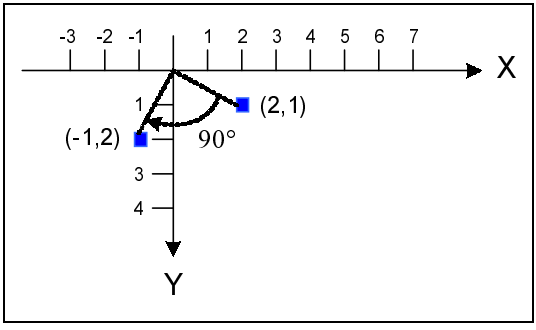


Рис. 5.24. Поворот на 90°

- Отражение относительно оси X (рис. 5.25).

$$\begin{pmatrix}2 & 1\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}=\begin{pmatrix}2 & -1\end{pmatrix}$$

Все преобразования, показанные на приведенных рисунках, являются линейными.

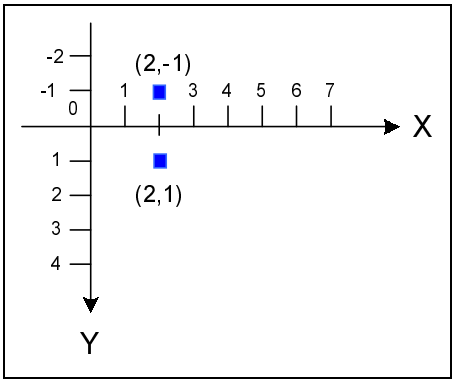


Рис. 5.25. Отражение относительно оси X

В то же время другие преобразования, такие как трансляция, не являются таковыми и не могут быть выполнены путем умножения на матрицу 2×2. Положим, мы имеем точку (2, 1) и хотим повернуть ее на 90°, затем транслировать (сдвинуть) на 3 единицы вдоль оси X и на 4 единицы вдоль оси Y. Мы можем выполнить это преобразование, сначала выполнив умножение, затем сложение.

На рис. 5.26 показан пример такого преобразования:

$$\begin{pmatrix}2 & 1\end{pmatrix}\begin{pmatrix}0 & 1\\ -1 & 0\end{pmatrix}+\begin{pmatrix}3 & 4\end{pmatrix}=\begin{pmatrix}2 & 6\end{pmatrix}$$

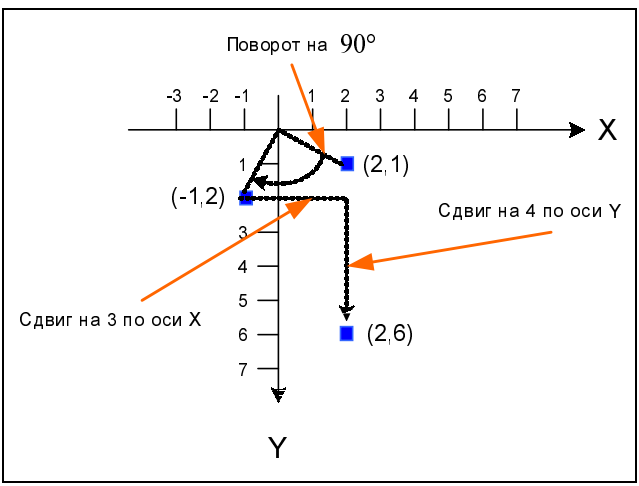


Рис. 5.26. Аффинное преобразование

Линейное преобразование (умножение на матрицу 2×2), за которым следует трансляция (добавление матрицы 1×2), называется *аффинным преобразованием*.

Альтернативой к существующему аффинному преобразованию (одно линейное преобразование и одна трансляция) служит полное преобразование с помощью матрицы 3×3. Для осуществления этого преобразования точка на плоскости должна быть представлена матрицей 1×3, в которой имеется третья, фиктивная, координата, обычно принимаемой равной 1.

Например, точка (2,1) представляется в виде матрицы $\begin{bmatrix}2 & 1 & 1\end{bmatrix}$.

Следующий пример показывает, как выполняется аффинное преобразование из предыдущего примера путем умножения на матрицу 3×3.

$$\begin{pmatrix}2 & 1 & 1\end{pmatrix}\begin{pmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 3 & 4 & 1\end{pmatrix}=\begin{pmatrix}2 & 6 & 1\end{pmatrix}$$

Заметим, что третья колонка содержит числа 0, 0, 1. Это случай матрицы 3×3 для аффинного преобразования. Значащими элементами в этой матрице являются элементы в первых двух столбцах. Верхний левый угол матрицы 3×3 фактически содержит матрицу 2×2 линейного преобразования, а первые два элемента 3-й строки трансляционную матрицу 1×2.

Разделение вы можете увидеть на рис. 5.27.

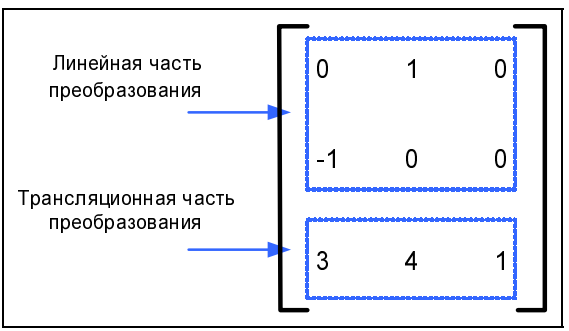


Рис. 5.27. Разделение матрицы

В GDI+ можно хранить аффинные преобразования в объекте `Matrix`. Поскольку третья колонка матрицы (0, 0, 1) фиктивная, необходимо иметь только 6 чисел при создании объекта `Matrix`.

Вызов

```
myMatrix := TGPMatrix.Create(0.0, 1.0, -1.0, 0.0, 3.0, 4.0);
```

создает матрицу, указанную в предыдущем примере.

Композитные (составные) преобразования

Композитным (составным) преобразованием является последовательность преобразований, следующих одно за другим. Рассмотрим следующие матрицы и выполняемые ими преобразования:

- матрица A — осуществляет поворот на 90°;
- матрица B — растягивает в 2 раза вдоль оси X;
- матрица C — транслирует на три единицы вдоль оси Y.

Возьмем точку (2, 1), представленную матрицей $\begin{bmatrix}2 & 1 & 1\end{bmatrix}$. Последовательное умножение ее на A, затем на B, затем на C подвергнет точку трем указанным преобразованиям:

$$\begin{bmatrix}2 & 1 & 1\end{bmatrix} ABC = \begin{bmatrix}-2 & 5 & 1\end{bmatrix}$$

Вместо того чтобы держать все три преобразования в трех отдельных матрицах, можно сначала последовательно выполнить их умножение, чтобы получить одну матрицу 3×3, которая будет содержать композитное преобразование все целиком. Положим, $ABC = D$. Тогда все преобразование будет выглядеть следующим образом:

$$\begin{bmatrix}2 & 1 & 1\end{bmatrix} D = \begin{bmatrix}-2 & 5 & 1\end{bmatrix}$$

Вот как будет выглядеть последовательное произведение матриц A, B и C:

$$\underset{A}{\begin{pmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}}\underset{B}{\begin{pmatrix}2 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}}\underset{C}{\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 3 & 1\end{pmatrix}} = \underset{D}{\begin{pmatrix}0 & 1 & 0\\ -2 & 0 & 0\\ 0 & 3 & 1\end{pmatrix}}$$

Из того факта, что матрица композитного преобразования может быть сформирована произведением отдельных преобразований, следует вывод, что любая последовательность аффинных преобразований может быть представлена в одном объекте `Matrix`.

> ***Внимание!***
>
> Последовательность умножения в композитном преобразовании имеет значение. В нашем случае поворот, затем масштабирование, затем трансляция — не то же самое, что масштабирование, затем поворот, затем трансляция, то есть *ABC* не равно *BAC*!

Класс `Matrix` снабжен различными методами для выполнения композитных преобразований: `Multiply`, `Rotate`, `RotateAt`, `Scale`, `Shear` и `Translate`.

Пример, представленный в листинге 5.22, создает матрицу композитного преобразования, которое сначала поворачивается на 30°, затем растягивается

в 2 раза вдоль оси Y и, наконец, смещается (транслируется) на 5 единиц вдоль оси X.

Листинг 5.22. Композитное преобразование

```
myMatrix := TGPMatrix.Create;
myMatrix.Rotate(30.0);
myMatrix.Scale(1.0, 2.0, MatrixOrderAppend);
myMatrix.Translate(5.0, 0.0, MatrixOrderAppend);
```

В результате выполнения композитных преобразований получится итоговая матрица:

$$\begin{pmatrix} \cos 30^\circ & 2\sin 30^\circ & 0 \\ -\sin 30^\circ & 2\cos 30^\circ & 0 \\ 5 & 0 & 1 \end{pmatrix} \approx \begin{pmatrix} 0{,}866 & 1 & 0 \\ 0{,}5 & 1{,}73 & 0 \\ 5 & 0 & 1 \end{pmatrix}$$

Глобальные и локальные преобразования (трансформации)

Глобальное преобразование — это преобразование, которое применимо к любому элементу, нарисованному конкретным объектом `Graphics`.

Чтобы построить глобальную трансформацию, нужно создать объект `Graphics`, затем вызвать метод `SetTransform`. Метод `SetTransform` оперирует объектом `Matrix`, который ассоциирован с объектом `Graphics`.

Преобразование, хранящееся в объекте `Matrix`, является мировым преобразованием.

Мировое преобразование может быть простым линейным преобразованием или сложной последовательностью аффинных преобразований. Несмотря на кажущуюся сложность, любое мировое преобразование хранится в одном объекте `Matrix`.

Класс `Graphics` располагает несколькими методами для построения композитных мировых преобразований: `MultiplyTransform`, `RotateTransform`, `ScaleTransform` и `TranslateTransform`.

При установке (включении) данных преобразований с помощью соответствующих методов последующий вывод различных графических объектов будет происходить с их учетом. Для сброса трансформаций следует воспользоваться методом `ResetTransform` класса `Grapchics`. Данный метод осуществляет глобальный сброс всех композитных преобразований и фактически установит настройки вывода по умолчанию. В терминах матричного исчис-

ления в этом случае композитная матрица принимает вид единичной диагональной матрицы:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Масштабирование

Установка масштабирования при выводе графических объектов GDI+ задается методом `ScaleTransform`. Коэффициенты масштабирования задаются в двух параметрах `SX` и `SY`. Параметр `SX` метода `ScaleTransform` принимает коэффициент масштабирования по горизонтали. Параметр `SY` — по вертикали.

Работу преобразования можно объяснить на следующем примере. Допустим, вы нарисовали сложный полигон, далее в работе вам нужно увеличить данный полигон в 2 раза. Это можно сделать, перемножив значения X и Y координат всех опорных точек в два раза, после чего заново произвести вывод по новым координатам. Согласитесь, это не совсем удобно, гораздо проще воспользоваться методом трансформации `ScaleTransform` и установить коэффициенты увеличения `SX` и `SY` равными двум. После чего произвести вывод полигона с теми же координатами.

В результате данный полигон будет нарисован автоматически увеличенным в 2 раза. На рис. 5.28 представлен результат включения данного преобразования:

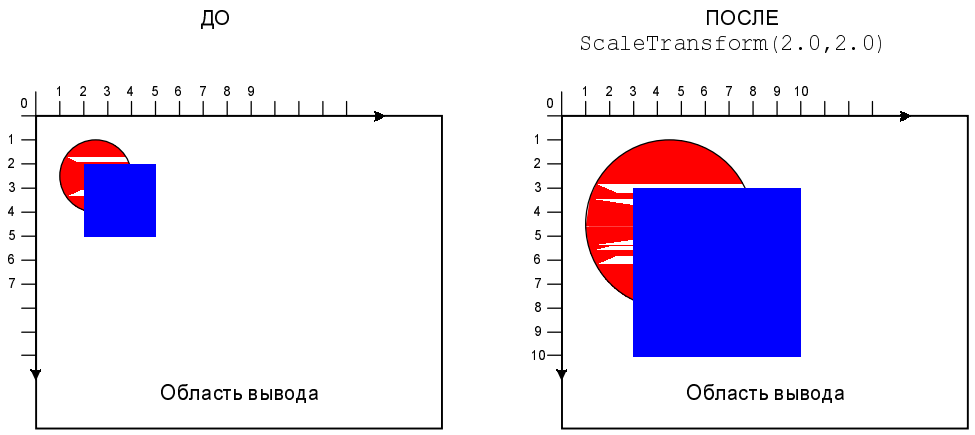


Рис. 5.28. Использование `ScaleTransform`

Вращение (поворот)

Установка поворота страничного координатного пространства при выводе графических объектов GDI+ задается методом `RotateTransform`. Угол поворота в градусах задается в параметре `Agle`. После включения данной трансформации страничная координатная система разворачивается на заданный угол.

На рис. 5.29 представлен результат включения данного преобразования.

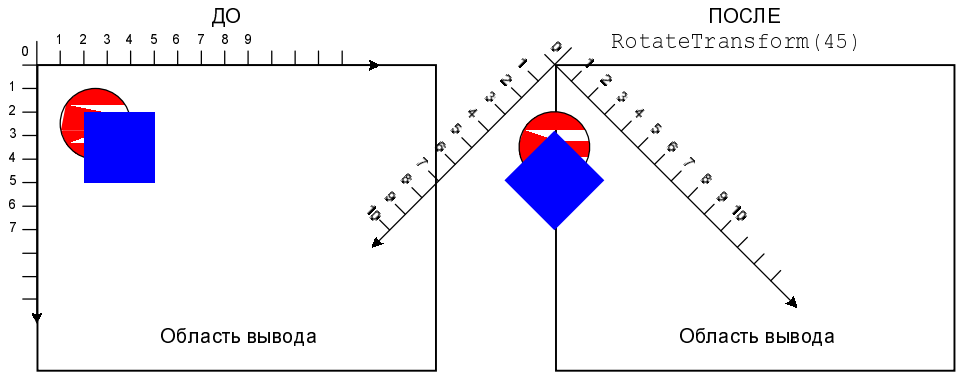


Рис. 5.29. Использование `RotateTransform`

Сдвиг

Установка сдвига страничного координатного пространства при выводе графических объектов GDI+ задается методом `TranslateTransform`. Сдвиг по оси X задается параметром `DX`, по оси Y — `DY`.

На рис. 5.30 вы видите результат сдвига координатного пространства.

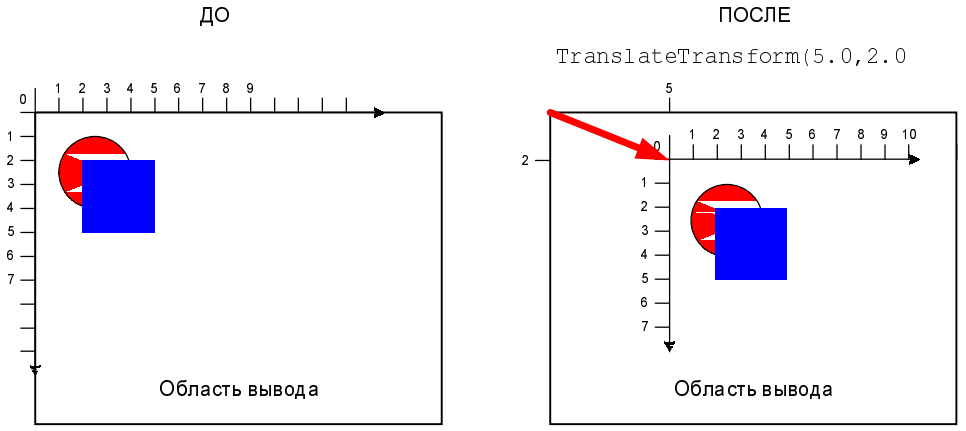


Рис. 5.30. Использование `TranslateTransform`

Пример, демонстрирующий использование мировых преобразований GDI+, вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05 \Ex29.

Использование регионов

Графический объект "регион" (в некоторых источниках "region" переводят как "область") представляет собой плоскую произвольную область. Стоит отметить, что регионы были еще в старом GDI, однако в GDI+ они претерпели ряд существенных изменений: регионы теперь не привязаны к координатам устройства, а также стали подчиняться координатным преобразованиям. Библиотека GDI+ позволяет создавать регионы любой формы и степени сложности. Однако за это приходится платить относительно большой ценой — памятью, а в случае применения региона для создания окон — процессорным временем при перерисовке окна.

Применение регионов достаточно разнообразно:

- создание непрямоугольных окон;
- создание области для ограничения отрисовки (clipping) — применяется для различных графических эффектов, а также в 2D-графике. Задав регион области вывода, можно добиться интересных эффектов. Например, создав сложный регион в виде звезды, вы можете вывести различный текст, при этом части текста, выходящие за пределы региона, будут обрезаны, как и другие графические объекты. Пример такого использования регионов представлен на рис. 5.31;

Рис. 5.31. Использование регионов для обрезки

- определение принадлежности произвольной точки региону — данная проверка часто используется при создании различных игр, собственных графических интерфейсов, а также применяется в 2D-графике;
- применение регионов в оформительских целях.

Для создания регионов в GDI+ предусмотрен класс `Region`. Стоит также отметить, что сложные регионы в свою очередь могут состоять из простых регионов, объединенных вместе. При объединении регионов могут использоваться различные методы объединения, представленные в классе `Region`, такие как:

- `Intersect` — пересечение, при объединении 2-х регионов конечный регион примет форму области пересечения;
- `Union` — объединение, конечный регион примет форму двух регионов;
- `Xor` — исключение пересечения, конечный регион примет форму двух регионов, за исключением области пересечения;
- `Exclude` — исключение по границе второго региона, конечный регион примет форму первого региона, обрезанного границей второго;
- `Complement` — исключение по границе первого региона, конечный регион примет форму второго региона, обрезанного границей первого.

Примеры методов объединения регионов представлены на рис. 5.32.

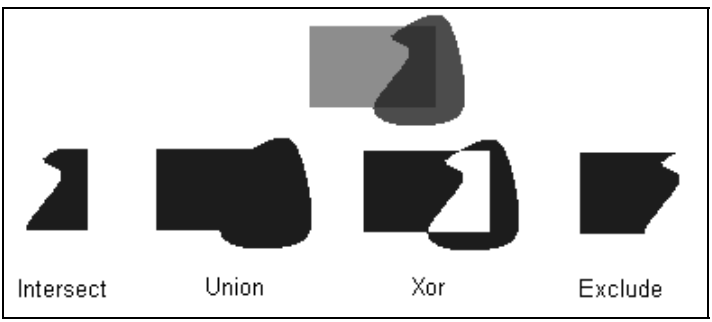


Рис. 5.32. Использование регионов для обрезки

Для демонстрации работы методов объединения регионов мы создали небольшую программу, которую вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex30.

Создание области для ограничения отрисовки

Один из самых частых примеров использования регионов является отрисовка (clipping). Для задания региона в качестве области вывода используется метод `SetClip` класса `Graphics`. Для снятия ограничивающей области следует вызвать метод `ResetClip`.

Для показа возможностей Clipping мы немного переработали программу, представленную в *разд. "Использование регионов"*. Полный исходный код этой программы вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex31.

Определение принадлежности произвольной точки региону

Для определения принадлежности координат точки какой-либо области, нужно воспользоваться универсальным методом `IsVisible`, представленным в классе `Grapchics`. Для определения принадлежности данной точки конкретному региону необходимо вызвать перегруженный метод, представленный в классе-потомке `Region`.

Для демонстрации метода `IsVisible` мы создали небольшую программу, исходный код которой вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex32.

Печать

Ни одно современное приложение не обходится без функции печати. Неважно, чем вы занимаетесь, — разрабатываете базы данных или создаете другие приложения, в большинстве случаев вам требуется печать. Встроить возможность печати в ваши программы, разработанные с использованием GDI+, несложно. Для этого достаточно выполнить ряд определенных действий при графическом выводе. Сначала вам нужно получить хендл (Handle) нужного принтера для задания его в качестве контекста устройства вывода в конструкторе класса `Graphics`. Для инициализации принтера вы вызываете метод WinAPI `StartDoc` с дескриптором принтера и указателем на структуру `TDocInfo`. Затем вы размещаете код, производящий графический вывод, на странице между методами `StartPage` и `EndPage`. После чего совершаете непосредственно отправку на печать, используя метод `EndDoc`.

Для получения дескриптора принтера, установленного по умолчанию, можно воспользоваться свойством `Handle` объекта `TPrinter` в Delphi. Однако очень часто пользователь сам перед отправкой на печать должен выбрать нужный ему принтер. Для предоставления такой возможности можно использовать пример, представленный в листинге 5.23. Используя стандартное диалоговое окно выбора принтера WinAPI, пользователь выбирает нужный ему принтер, а программист получает нужный дескриптор, возвращаемый в `Pd.hDc`.

Листинг 5.23. Получение дескриптора принтера

```
procedure TForm1.Button1Click(Sender: TObject);
var
  PrintHandle: Thandle  ; // Дескриптор (хендл) принтера
  PD         : TPrintDlg;
begin
  FillChar(Pd, Sizeof(PD), #0); // Обнуляем структуру
```

```
  Pd.lStructSize := sizeof(Pd); // Размер
  Pd.hWndOwner := Form1.Handle; // Владелец окна
  Pd.Flags := PD_RETURNDC;   // Получаем хендл принтера
  IF PrintDlg(pd) then begin
     PrintHandle := Pd.hDc;
     ShowMessage(Format('Хендл принтера - $%x',[PrintHandle]));
  end;
end;
```

После получения дескриптора принтера программист должен заполнить структуру WinAPI `TDocInfo`, предназначенную для передачи информации о печатаемом документе, на спуллер печати. Структура `TDocInfo` имеет следующие элементы для заполнения, представленные в табл. 5.19.

Таблица 5.19. *Элементы структуры* `TDocInfo`

Элемент	Описание
`cbSize`	Устанавливает размер структуры в байтах. Дело в том, что в различных версиях Windows размер структуры разный. Для получения правильного размера структуры нужно воспользоваться функцией `SizeOf`: `docInfo.cbSize := sizeof(DOCINFO);`
`lpszDocName`	Имя документа, выводимого на печать
`lpszOutput`	Имя выходного файла, если вывод происходит в файл. Если значение этого параметра установлено в `Null`, вывод будет направлен на устройство обозначенным параметром `Hdc`, который передан функции `StartDoc`. Этот параметр не является обязательным
`fwType`	Дополнительная информация о задании на печать. Этот параметр не является обязательным

После того как структура `TDocInfo` будет заполнена, нужно произвести запуск задания на печать. Делается это с помощью функции WinAPI `StartDoc`, где в качестве параметра `Hdc` передается дескриптор принтера, а в качестве параметра `DocInfo` — структура `TDocInfo`.

Начало и конец страницы для вывода устанавливается функциями WinAPI `StartPage` и `EndPage` с параметром `Hdc`, указывающим дескриптор принтера. Функция `StartPage` сообщает драйверу принтера о начале приема данных для новой страницы. В приложении необходимо вызывать эту функцию после функции `EndPage` каждый раз, когда во время печати происходит переход на новую страницу.

Функция WinAPI `EndDoc` заканчивает задание на печать, начатое функцией `StartDoc`. Обычно после вызова данной функции начинается печать на принтере. Данная функция в качестве параметра `Hdc` принимает все тот же дескриптор принтера.

Шаблон кода, производящего необходимые установки и выполняющего печать одного листа, представлен в листинге 5.24.

Полный исходный код примера, производящего вывод на принтер, вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex33.

Листинг 5.24. Печать из GDI+

```
procedure TForm1.Button1Click(Sender: TObject);
var
 PrintHandle  : Thandle ;  // Дескриптор (хендл) принтера
 PD      : TPrintDlg;  // Окно выбора
 docInfo    : TDOCINFO;   // Информация о печатаемом документе
begin
 FillChar(Pd, Sizeof(PD), #0); // Обнуляем структуру
 Pd.lStructSize := sizeof(Pd); // Размер
 Pd.hWndOwner := Form1.Handle; // Владелец окна
 Pd.Flags := PD_RETURNDC;   // Установка флага - получать
                // Хендл принтера
 IF PrintDlg(pd) then
 begin
   PrintHandle := Pd.hDc;
   ZeroMemory(@docInfo, sizeof(DOCINFO)); // Чистим память
   docInfo.cbSize   := sizeof(DOCINFO);
   docInfo.lpszDocName := 'Печать GDI+'; // Имя документа

   StartDoc(PrintHandle, docInfo); // Инициализация перед отправкой на
                  // печать - начало задания печать
   StartPage(PrintHandle);     // Начало страницы печати
   // Вывод на принтер
   graphicsGDIPlus := TGPGraphics.Create(PrintHandle);
   // * * * * * * * * * * * * * * * * * * * *
   // * ДАЛЕЕ ПРОИСХОДИТ ГРАФИЧЕСКИЙ ВЫВОД*
   // * * * * * * * * * * * * * * * * * * * *
   // Не забываем самое главное - освободить память
   graphicsGDIPlus.Free;
   // Конец страницы и печати
   EndPage(PrintHandle);
   EndDoc(PrintHandle);
 end;
end;
```

Стоит также заметить, что при выводе на печать очень часто нужно узнать различные характеристики принтера, такие как размеры полей, разрешение печати, формат и пр. Данные характеристики требуются для соответствующей настройки графического вывода, установки соответствующих коэффициентов между экраном и принтером.

Получить данные характеристики несложно, используя функцию запроса сведений о возможностях устройства WinAPI GetDeviceCap. Пример получения характеристик принтера вы можете найти на прилагаемом компакт-диске в каталоге Source\Ch05\Ex34.

Заключение

В данной главе мы сделали попытку немного приоткрыть перед читателями мощные возможности нового графического ядра Microsoft, сделать их более доступными для рядовых программистов Delphi и, в какой-то мере, заставить их "сойти" со старого GDI, обретя мощь нового.

ГЛАВА 6

Windows API

Windows API — это набор функций интерфейса прикладного программирования, предоставляемых операционной системой каждой программе. Эти функции находятся в стандартных динамически компонуемых библиотеках (DLL), таких как kernel32.dll, user32.dll, gdi32.dll. Сами библиотеки расположены в каталоге %SystemRoot%\System. Так как 32-разрядная операционная система пришла на смену 16-разрядной, то одновременно произошла и замена 16-разрядного Windows API (Win16 API) на новый 32-разрядный вариант (Win32 API). Поэтому нужно иметь в виду, что за небольшим исключением набор Win32 API является единым для семейств операционных систем Windows 9*x* и Windows NT. По области действия функции Windows API (или просто API) можно разбить на несколько групп. В данной главе приведены краткие описания наиболее используемых стандартных функций Windows API, работающих в 32-разрядных операционных системах Windows 98/2000/XP, и рассматривается на примерах работа многих из них.

Типы данных

Windows API использует множество типов данных: для одних в Delphi существуют точные аналоги, для других приходится использовать совместимые типы. В табл. 6.1 представлены наиболее общие типы и их аналоги в Delphi.

Таблица 6.1. *Типы данных Windows API и их аналоги в Delphi*

Идентификатор типа в Windows	Тип, используемый в Delphi	Описание идентификатора
ATOM	Word	Определяет индекс строки для локальной или глобальной таблицы атомов
BOOL	LongBool	Логический тип

Таблица 6.1 *(продолжение)*

Идентификатор типа в Windows	Тип, используемый в Delphi	Описание идентификатора
COLORREF	DWORD	Определяет значение цвета в формате RGB
DWORD	LongWord	Целое число
FARPROC	Pointer	Указатель на процедуру, которая обычно выступает в качестве параметра в функциях, использующих функции обратного вызова
HANDLE	THandle	Определяет дескриптор какого-либо объекта
HBITMAP	LongWord	Манипулятор графического объекта, представляющего собой область памяти для хранения изображения
HBRUSH	LongWord	Манипулятор объекта "кисть", которым закрашиваются фигуры во время рисования
HCURSOR	HICON	Манипулятор указателя мыши. Им служит изображение размером до 32×32 пикселов
HDC	LongWord	Манипулятор контекста устройства
HENHMETAFILE	LongWord	Определяет дескриптор объекта расширенного метафайла
HFILE	LongWord	Определяет манипулятор объекта файла на диске
HFONT	LongWord	Определяет манипулятор объекта шрифта
HGDIOBJ	LongWord	Дескриптор объекта GDI
HGLOBAL	THandle	Определяет дескриптор, идентифицирующий глобально размещенный в памяти объект. (В 32-разрядных версиях Windows разница между глобально и локально размещенными в памяти объектами отсутствует)
HHOOK	LongWord	Дескриптор установленной в системе hook-точки
HICON	LongWord	Определяет манипулятор объекта иконки
HINST	THandle	Определяет манипулятор экземпляра работающего объекта Windows
HKL	LongWord	Дескриптор раскладки клавиатуры
HLOCAL	THandle	Определяет дескриптор, идентифицирующий локально размещенный в памяти объект. (В 32-разрядных версиях Windows разница между глобально и локально размещенными в памяти объектами отсутствует)

Таблица 6.1 *(продолжение)*

Идентификатор типа в Windows	Тип, используемый в Delphi	Описание идентификатора
HMENU	LongWord	Определяет манипулятор объекта меню или всплывающего меню
HMETAFILE	LongWord	Определяет манипулятор объекта метафайла
HMODULE	HINST	Определяет манипулятор программного модуля (например, DLL или модуля приложения). Часто используется для обращения к ресурсам внутри модуля (шрифтам, значкам, указателям мыши и т. д.)
HPALETTE	LongWord	Определяет манипулятор цветовой палитры Windows
HPEN	LongWord	Определяет тип линии объекта при прорисовке
HRGN	LongWord	Определяет тип описателя участка окна. Обычно используется для определения областей отсечения (частей окна, в которых допускается графический вывод)
HRSRC	THandle	Определяет манипулятор объекта ресурса
HSTR	THandle	Определяет тип для описателей строки
HWND	LongWord	Определяет манипулятор окна
INT	Integer	Определяет 32-разрядное знаковое целое число
LANGID	Word	Определяет идентификатор языка
LCID	DWORD	Локальный идентификатор
LONG	LongInt	Определяет 32-разрядное знаковое целое число
LPARAM	Longint	Параметр 32-битного сообщения
LPCSTR	PAnsiChar	Строковый указатель
LPDWORD	PDWORD	Указатель на 32-битное значение
LPHANDLE	PHandle	Определяет длинный указатель на описатель. Обычно не используется ObjectWindows, а включен для совместимости с кодами Windows, написанными на других языках
LPPOINT	TPoint	Определяет координаты точки на экране или в окне
LPSTR	PChar	Указатель на строковую константу с нулем в конце 8-разрядных символов (ANSI) Windows

Таблица 6.1 *(продолжение)*

Идентификатор типа в Windows	Тип, используемый в Delphi	Описание идентификатора
LPSTR	PAnsiChar	Строковый указатель
LPVOID	Pointer	Определяет длинный указатель. Обычно не используется ObjectWindows, а включен для совместимости с кодами Windows, написанными на других языках
LPWSTR	PWideChar	То же, что и LPSTR, но для Unicode
LRESULT	Longint	32-битное значение, возвращаемое функцией
MAKEINTATOM	PStr	Используется для приведения целых чисел к атомам
MAKEINTRESOURCE	PStr	Используется для приведения целых чисел к именам ресурсов
PBOOL	^BOOL	Определяет указатель на логическое значение
PBYTE	^Byte	Определяет указатель на 8-битное значение без знака
PDOUBLE	^Double	Определяет указатель на вещественное значение двойной точности
PDWORD	^DWORD	Определяет указатель на 32-битное значение
PHANDLE	^THandle	Определяет указатель на манипулятор
PINT	^Integer	Определяет указатель на 32-битное значение
PINTEGER	^Integer	Определяет указатель на 16-битное целое число со знаком
PLONGINT	^Longint	Определяет указатель на 32-битное целое число
PSINGLE	^Single	Определяет указатель на вещественное значение одинарной точности
PSMALLINT	^Smallint	Определяет указатель на 16-битное значение
PSTR	PChar	Определяет указатель на нуль-терминальную строку
PUCHAR	^Byte	Определяет указатель на 8-битное значение
PUINT	^UINT	Определяет указатель на беззнаковое 32-битное целое число
PULONG	^ULONG	Определяет указатель на беззнаковое 32-битное целое число

Таблица 6.1 *(окончание)*

Идентификатор типа в Windows	Тип, используемый в Delphi	Описание идентификатора
PVOID	Pointer	Указатель на тип
PWORD	^Word	Определяет указатель на 16-битное число без знака
SHORT	Smallint	Знаковое 16-битное целое число
THANDLE	LongWord	Определяет общий тип дескриптора
TATOM	Word	Определяет указатель на 16-битное сообщение, определяющее атом или пересылаемое между DDE-приложениями
UCHAR	Byte	8-битное значение (может также содержать символ)
UINT	LongWord	Беззнаковое 32-битное целое число
ULONG	Cardinal (Longint)	Беззнаковое 32-битное целое число
WCHAR	WideChar	Определяет Unicode-символ
WPARAM	Longint	32-битный параметр сообщения. В ранних версиях Windows он 16-битный

Константы

Функции Windows API содержат тысячи различных констант, используемых в качестве параметров. В файле Windows.pas можно найти много информации относительно констант той или иной функции. Не будет лишним чаще обращаться к тексту данного файла, особенно при работе со сложными функциями.

Примечание

Лучше заранее сделать копию этого файла и пользоваться ей в работе, чтобы избежать случайного изменения в рабочем файле.

Строки

Так как операционная система Windows изначально была написана на языке C++, это наложило определенные ограничения на программистов Delphi. Дело в том, что паскалевский строковой тип `String` Windows API не известен,

вместо него используются символьные массивы (признаком конца строки служит символ с кодом нуль), поэтому для совместимости с такими строками в Delphi, начиная с версии 3, введен тип "PChar".

Физически тип `PChar` является просто указателем на символ (`^char`), однако, если присмотреться к нему внимательней, у него гораздо больше свойств, чем у простого указателя. Во-первых, переменная типа `PChar` может рассматриваться как массив символов с терминальным нулем, т. е., если вам нужно передать в некоторую функцию переменную типа `PChar`, вы можете пойти двумя путями:

1. Использовать массив:

```
var
  vParametr : array[0..255] of char;

APIFunc(vParametr); // Вызов функции и передача массива символов
```

2. Выделить область памяти и передать указатель на нее в функцию:

```
var
  vParametr : PChar;

GetMem(vParametr,256);
APIFunc(vParametr);
FreeMem(vParametr,256);
```

Приведенные выше примеры эквивалентны по своему действию, однако первый вариант используется обычно в случае, когда длина передаваемой строки не превосходит некоторое небольшое число; а второй — когда размер получаемой строки неизвестен на момент объявления переменной.

Стоит также заметить, что вы можете работать с переменной типа `PChar`, как и с обычным массивом. Например, вы можете извлекать нужный символ из нее так, как если бы извлекали элемент массива:

```
ch := p[10];
```

Для работы с типом `PChar` в Delphi существует ряд функций, начинающихся с префикса `Str` (`StrPas` и т. д.), полный их список можно увидеть в справке по Delphi. Все эти функции расположены в модуле sysutils.

Для преобразования обычных паскалевских строк в тип `PChar` в Delphi предусмотрена функция `StPCopy`, однако иногда удобнее использовать следующий трюк:

```
Var
  vParametr : String;
```

```
Begin
  VParametr :='Example';
  APIFunc(@vParametr [1]);
End;
```

Мы просто передаем в процедуру адрес первого символа текстовой строки, при этом конечный нулевой символ уже существует в конце string-строки. Для обратного преобразования существует функция `StrPas`, однако допустимо и простое присваивание string-строке строки, завершающейся терминальным нулем.

Дескрипторы

Одним из важнейших понятий в Windows-программировании является понятие объектного *дескриптора* (handle). В Windows все объекты — кисти, перья, растры, указатели мыши, контексты устройств, окна, экземпляры программ — идентифицируются 32-разрядным (в Win16 — 16-разрядным) целым числом, которое называется *дескриптором* (иногда *манипулятором*). С каждым дескриптором связывается идентификатор типа, начинающийся с буквы "H" (в языке Си — с "h" нижнего регистра. В отличие от Pascal, регистр в Си имеет значение). Например, `hWnd` — это дескриптор окна. Дескрипторы ссылаются на объекты, находящиеся под управлением системы Windows. Работающая операционная система отслеживает все дескрипторы, которые служат связующими звеньями между загруженными объектами и вызвавшими их приложениями.

Когда мы создаем некоторый объект в Windows, ему присваивается уникальный 32-разрядный дескриптор. В дальнейшем, при работе с этим объектом, каждой функции передается этот дескриптор. В этом заключается главное различие между стандартными методами класса Object Pascal и функциями Windows API. Первые связаны с тем экземпляром класса, через который они вызываются, и поэтому не требуют явного указания на объект. Вторым необходимо такое указание (что и делается с помощью дескриптора), т. к. они сами по себе никак не связаны ни с одним объектом.

В Delphi для хранения дескриптора объектов определен тип `THandle`:

```
type THandle : LongInt;
```

Кроме типа `THandle` в Delphi поддерживаются также имена типов, пришедшие из C++, а именно: `HWND`, `HMENU`, `HKEY` и другие имена, начинающиеся с префикса H (Handle).

При использовании функций Windows API к объектам Object Pascal следует понимать, что данные функции "не знают" о внутренних механизмах работы Delphi.

Поясним на примере: если спрятать окно не с помощью метода Delphi — `Hide`, а с помощью вызова функции Windows API `ShowWindow(Handle, SW_Hide)`, то в Delphi не возникнет событие `OnHide`, потому что оно генерируется упомянутыми внутренними механизмами. Следует об этом помнить.

Но такие недоразумения случаются обычно только тогда, когда функциями Windows API дублируется то, что можно сделать и с помощью Delphi. Для вызова функций Windows API для объекта, созданного с помощью Delphi, используйте свойство объекта `Handle`, — в нем хранится дескриптор.

Сообщения

Сообщения операционной системы Windows очень напоминают событийные сообщения Delphi, реализованные в VCL-объектах. Схема использования сообщений Windows такая же — программист определяет реакцию (пишет код обработчика события) на определенное событие. По мере наступления события операционная система передает управление в данный обработчик.

В отличие от событий VCL, где уже предопределены основные обработчики событий в объектах (программист только пишет код обработки в определенном обработчике соответствующего объекта), сообщения (message) операционной системы не имеют в программе явно выраженных обработчиков. Хотя данное утверждение не совсем верно, т. к. на нижнем уровне объектов VCL эти обработчики прописаны явно, можно сказать, что это так. В приложение поступает огромное количество сообщений Windows, какие-то из них обрабатываются на уровне объектов VCL, но большинство не обрабатывается вообще. То есть, если вы хотите обработать какое-либо специфичное сообщение Windows, обработка которого не предусмотрена в объектах VCL Delphi, вы должны явно создать собственный обработчик сообщений.

Сообщения Windows делятся на следующие:

- сообщения, предоставляющие информацию, — данный вид посылает запрос какому-либо элементу для получения некой информации. Например, когда Windows хочет узнать заголовок окна, он посылает этому окну специальное сообщение, в ответ на которое окно должно сообщить системе свой заголовок;
- уведомляющие сообщения — данный вид уведомляет приложение о начале какого либо действия и предоставляет возможность вмешаться в него. Например, в состав таких сообщений входит уведомление программы о начале перетаскивания окна.

В Delphi для реакции на каждое событие обычно создается свой метод. В Windows одна процедура, называемая оконной, обрабатывает все сообщения.

Каждое сообщение Windows имеет свой уникальный номер (идентификатор), а оконная процедура целиком состоит из оператора `case`, и каждому сообщению соответствует своя альтернатива этого оператора.

Уникальные номера сообщений и их константы можно найти в модуле Messages.pas. Представленные там константы начинаются с префикса, указывающего на принадлежность сообщения к определенной группе. Например, сообщения общего назначения начинаются с `WM_`: например, `WM_Paint`, `WM_GetTextLength`, а сообщения для кнопок начинаются с префикса `BM_`.

Помимо уникального идентификатора каждое сообщение содержит два параметра: `WParam` и `LParam`.

Буквы `W` и `L` ("Word" и "Long") означают: первый параметр 16-разрядный, а второй — 32-разрядный.

Однако так было только в старых, 16-разрядных версиях Windows. В 32-разрядных версиях оба параметра 32-разрядные, несмотря на их названия.

Конкретный смысл каждого параметра зависит от сообщения. В некоторых сообщениях один или оба параметра могут вообще не использоваться, в других, наоборот, двух параметров не хватает. В этом случае один из параметров (обычно `LParam`) содержит указатель на дополнительные данные.

Кроме параметров `WParam` и `LParam`, каждому сообщению приписывается время возникновения и координаты курсора в момент возникновения. Эти параметры можно узнать с помощью функций `GetMessageTime` и `GetMessagePos` соответственно.

После обработки сообщения оконная процедура должна вернуть какое-то значение. Обычно это значение просто сигнализирует, что сообщение не нуждается в дополнительной обработке, но в некоторых случаях оно бывает более осмысленно, например, `WM_SetIcon` должно вернуть дескриптор иконки, которая была установлена ранее. Если программист не хочет обрабатывать сообщение самостоятельно, он должен вызвать для его обработки функцию `DefWindowProc`.

Следует также заметить, что все поступающие сообщения помещаются в так называемую очередь сообщений. Очередь сообщений своя для каждой нити. Нить должна сама выбирать сообщения из этой очереди, транслировать их и затем вызывать функцию `DispatchMessage`, чтобы направить это сообщение в нужную оконную процедуру.

На практике обычно все эти сложности не требуются, зачастую нужно лишь написать обработчик на какое-либо нестандартное сообщение.

Как вы знаете, Delphi предоставляет программисту все средства, необходимые для обработки сообщений. Простейший способ обработки сообщений заключается в использовании директивы `message`.

На практике это выглядит примерно так (ниже представлена обработка сообщения WM_MOVE).

```
type TSomeForm = class(TForm)
          { процедура обработки сообщения Windows}
          procedure WMMOVE(var Message: TMessage); message
                    WM_MOVE;
procedure TSomeForm.WMMOVE;
 begin
  {что-то делаем}
  inherited
 end;
```

Механизм работы оконной процедуры в Delphi устроен так, что система сама ищет среди методов класса специальные методы для обработки сообщения Windows, использующие директиву message.

Как вы уже заметили, в отличие от обработчиков VCL для процедуры обработки сообщения Windows имя метода не имеет значения, значение имеет только константа, уникально идентифицирующая сообщение и стоящая после директивы message. Именно поэтому при вызове перекрытого метода для обработки данного сообщения достаточно написать inherited без указания имени метода. Такой способ вызова не приведет к ошибке даже в том случае, если класс-родитель вообще не имел метода для обработки такого сообщения.

Само сообщение представлено в Delphi типом TMessage:

```
  type
  TMessage = record
    Msg: Cardinal;
       case Integer of
         0: (WParam: Longint;  LParam: Longint;  Result: Longint);
         1: (WParamLo: Word;  WParamHi: Word;
             LParamLo: Word;  LParamHi: Word;
             ResultLo: Word;  ResultHi: Word);
       end;
```

Данный тип представляет собой запись, содержащую 32-разрядные целые поля Msg, WParam, LParam и Result.

Первое поле содержит уникальный идентификатор сообщения, два следующих — параметры сообщения, а в поле Result метод должен присвоить то значение, которое потом вернет системе данный обработчик. Именно из-за необходимости передавать значение параметр метода обработки сообщения должен быть переменной.

При обработке сообщений часто приходится сталкиваться с ситуациями, когда один 32-разрядный параметр используется для передачи двух 16-разрядных значений. Для облегчения работы тип `TMessage` описан как вариантная запись, поэтому в нем есть поля `WParamLo`, `WParamHi`, `LParamLo`, `LParamHi`, `ResultLo` и `ResultHi`, имеющие тип `Word` и дающие доступ к старшему и младшему словам соответствующего параметра.

Стоит также заметить, что тип `TMessage` — не единственный тип, который может иметь параметр метода обработки сообщения. Стоит сказать, что данный тип является наиболее часто используемым (универсальным) типом, однако для многих сообщений в модуле Messages.pas описаны собственные типы.

Их названия образованы от названия соответствующих сообщений. Например, для сообщения `WM_Paint` описан тип `TWMPaint`, для `WM_GetText` — `TWMGetText`.

В этих структурах все поля имеют тот тип, который наилучшим образом подходит для обработки именно этого сообщения. Кроме того, поля имеют названия, отражающие их назначение, что делает программу более удобной для чтения.

Но такие типы описаны не для всех сообщений, поэтому иногда приходится пользоваться универсальным `TMessage`.

Синтаксис функций Windows API

Большая часть функций API уже импортирована в среду Delphi и описана в ее модулях, их вызов практически ничем не отличается от вызова любых других функций Delphi. Если же функция отсутствует в Delphi, то ее можно импортировать из соответствующей DLL с помощью существующего механизма импорта функций *(см. разд. "Импортирование функций Windows API" в данной главе)*. Так как система Windows написана на Си, то все описания функций Windows API даны в DLL в соответствии с синтаксисом именно этого языка, а не языка Pascal. Для перевода функции в Delphi необходимо знать синтаксис описания функций в языке Си.

Описание функции в Си имеет вид:

```
<Тип функции> <Имя функции>(<Тип параметра> <Имя параметра>,
                             <Тип параметра> <Имя параметра>, ...);
```

В Си различаются верхний и нижний регистр, и идентификаторы HDC, hdc, hDC — для Си являются разными. Поэтому часто можно встретить фразу о том, что имя параметра и его тип совпадают с точностью до регистра. Конечно, в Pascal это уже не имеет значения.

Например, в Си функция:

```
HMETAFILE CopyMetaFile(HMETAFILE hmfSrc, LPCTSTR lpszFile);
```

в Delphi может иметь вид:

```
function CopyMetaFile(hmfSrc: HMETAFILE; lpszFile: LPCTSTR): HMETAFILE;
```

или

```
function CopyMetaFile(hmfSrc: HMetaFile; lpszFile: PChar): HMetaFile;
```

Несколько особняком стоит тип `VOID`. Если функция имеет такой тип, то в языке Pascal она описывается как процедура. Если вместо параметров у функции в скобках стоит `VOID`, это означает, что функция не имеет параметров. Например, Си-функция:

```
VOID CloseLogFile(VOID);
```

в Delphi описывается как:

```
procedure CloseLogFile;
```

> ***Примечание***
>
> Не путайте `VOID` и `PVOID`. `PVOID` — это нетипизированный указатель, соответствующий типу `Pointer`.

В тех случаях, когда тип параметра является указателем на другой тип (обычно начинается с букв LP), при описании этой функции в Delphi можно пользоваться параметром-переменной, т. к. функции передается указатель. Например, функция:

```
int GetRgnBox(HRGN  hrgn, LPRECT  lprc);
```

в файле Windows.pas описана так:

```
function GetRgnBox(RGN: HRGN; var p2: TRect): Integer;
```

Параметры функций

Подавляющее количество Windows API-функций использует переданные им статические параметры для выполнения с их участием каких-либо действий. Тем не менее определенная часть функций возвращает значения, которые должны быть сохранены в буфере, и этот буфер функция возвращает в виде указателя на него. Для большинства функций этот буфер должен быть создан самим приложением перед вызовом функции. Во многих случаях параметр может принимать одно или более значений из некоторой таблицы. Эти значения определяются как константы и объединяются с использованием логического оператора `OR`. Реальные параметры, передаваемые в функцию, иденти-

фицируются с битовой маской, в которой позиция определенных бит имеет определенный смысл для функции.

Пример такого слияния: функция CreateWindowEx имеет параметр, называемый dwStyle, который является суммой констант, связанных оператором OR. Чтобы передать функции более чем одну константу, параметр должен выглядеть, к примеру, так: "WS_CAPTION or WS_CHILD or WS_CLIPCHILDREN". (Будет создано дочернее окно с полосой заголовка, при этом будет запрещено рисование родительского окна в области, занятой дочерним окном.)

Наоборот, если функция должна возвращать одно или более значений, определяемых специальными константами, то с помощью логического умножения (оператор AND) определяется, входит ли результат в набор возвращаемых значений.

Импортирование функций Windows API

Если вызываемая функция API не описана в среде Delphi, то ее необходимо предварительно импортировать. Импортирование функций API в Delphi ничем не отличается от импортирования любых других функций из библиотек DLL.

Например, ниже представлено импортирование функции BroadcastSystemMessage:

```
interface
function BroadcastSystemMessage(Flags: DWORD; Recipients: PDWORD;
                                uiMessage: UINT; wParam: WPARAM;
                                lParam: LPARAM): Longint; stdcall;
. . . . . .
implementation
function BroadcastSystemMessage; external user32 name
                                        'BroadcastSystemMessage';
```

Любая функция Windows API может быть импортирована и использована в Delphi при условии, что известны ее параметры и DLL, содержащая эту функцию. Важно заметить, что директива stdcall должна быть добавлена в описание, т. к. она определяет стандартный для Windows механизм размещения параметров функции в стеке.

Нестандартно импортируемые функции

Некоторые функции импортируются в исходный код Delphi нестандартно. Они являются исключениями и указываются в описании конкретной функции. В большинстве случаев эти функции, имеющие дело с NIL в качестве па-

раметра типа "Указатель", возвращают реальные данные в буфер определенного размера, который должен быть создан заранее. В Delphi некоторые из этих функций импортируются с параметрами, объявляемые в VAR или CONST. Эти типы параметров могут быть указателями на буфер, но никогда не могут принимать значение Nil, что ограничивает их использование в Delphi.

Функции обратного вызова

Другое очень важное понятие в Windows-программировании — это понятие *функции обратного вызова* (callback-функции). Функцией обратного вызова называется функция приложения, которая никогда не вызывается напрямую другими функциями или процедурами этого приложения (хотя ничто не запрещает это делать), а вызывается операционной системой Windows. Это позволяет Windows общаться с приложением напрямую посредством различных параметров, определенных как функции обратного вызова. На эти функции налагаются следующие требования: во-первых, эти функции должны быть именно функциями, а не методами класса (хотя это иногда можно обойти); во-вторых, эти функции должны быть написаны в соответствии с моделью вызова stdcall.

В качестве примера такой функции может послужить функция EnumWindows.

В справочной системе она описана следующим образом:

```
BOOL EnumWindows(WNDENUMPROC  lpEnumFunc, LPARAM  lParam);
```

В Windows.pas она имеет вид:

```
function EnumWindows(lpEnumFunc: TFNWndEnumProc; lParam: LPARAM): BOOL;
                                                                stdcall
```

В качестве первого параметра должен стоять указатель на функцию обратного вызова. Синтаксис прототипа этой функции описан так (поскольку это только прототип, то реальное имя может быть любым):

```
BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM lParam);
```

Любым может быть и тип самой функции, и тип второго параметра, который разработчик может использовать по своему усмотрению, лишь бы его длина не превышала 32 бит.

Пример функции обратного вызова для случая, когда второй параметр имеет тип Longint, будет в Delphi выглядеть так:

```
function MyCallbackFunction(Wnd:HWnd; P: Longint):Bool; stdcall;
begin
  { что-то делается }
end;
```

```
var
  MyVar: Longint;

EnumWindows(@MyCallbackFunction, LongInt(MyVar));
```

Использование справочной системы по функциям Windows API

Функций Windows API так много, что запомнить все совершенно нереально, поэтому работа без справочника под рукой просто невозможна.

Наиболее доступным справочником для российского программиста является Win32 Developer's Reference, справочная система фирмы Microsoft, потому что фирма Inprise (тогда еще Borland) получила лицензию на включение ее в состав Delphi.

Наконец, если не удается понять, как функция, описанная в справке, должна быть переведена на Pascal, можно попытаться найти описание этой функции в исходных текстах модулей, поставляемых вместе с Delphi. Эти модули находятся в каталоге $(DELPHI)\Source\RTL\Win. Можно также воспользоваться подсказкой, которая всплывает в редакторе Delphi после набора имени функции.

Если посмотреть справку, например, по функции `GetSystemMetrics`, то видно, что функция должна иметь один целочисленный параметр. Однако далее в справке предлагается при вызове этой функции подставлять в качестве параметра не числа, константы `SM_ARRANGE`, `SM_CLEANBOOT` и т. д. Подобная ситуация существует и со многими другими функциями Windows API. `SM_ARRANGE`, `SM_CLEANBOOT` и т. д. являются именами числовых констант.

Эти константы описаны в том же модуле, что и функция, использующая их, поэтому можно не выяснять численные значения этих констант, а указывать при вызове функций их имена, например, `GetSystemMetrics(SM_Arrange)`. Если, по каким-то причинам, все-таки потребовалось выяснить численные значения, то в справочной системе их искать не стоит — их там нет. Могу только опять отправить к исходным текстам модулей Delphi, в которых эти константы описаны. Так, например, просматривая Windows.pas, можно узнать, что `SM_ARRANGE = 56`.

В описании многих функций Windows API можно увидеть три ссылки: `QuickInfo`, `Overview` и `Group`.

Первая дает краткую информацию о функции. Самой полезной частью этой информации является то, для каких версий Windows эта функция реализована. Например, очень полезна функция `MaskBlt`, однако `QuickInfo` показывает,

что она реализована только в Windows NT. Программа, использующая эту функцию, не будет работать в Windows 95. Иногда напротив названия одной из систем стоит слово "Stub", которое переводится как "пень", "обрубок" (например, для функции `GetDeviceGammaRamp` это слово стоит напротив Windows NT). Это означает, что в данной версии эта функция присутствует (т. е. обращение к ней не вызывает ошибки), но ничего не делает. Авторы оставляют на совести программистов из Microsoft вопрос, зачем нужны вот такие "пни".

`Overview` — это краткий обзор какой-то большой темы. Например, для любой функции, работающей с растровыми изображениями, обзор будет в двух словах объяснять, зачем в принципе нужны эти самые растровые изображения. По непроверенным данным, первоначально эти обзоры замышлялись, как нечто большее, но потом ограничились такими вот лаконичными фразами. Как бы то ни было, найти в обзоре полезную информацию удается крайне редко, поэтому заглядывать туда стоит, если, ну совсем ничего не понятно.

И, наконец, `Group`. Эта ссылка приводит к списку всех функций, родственных данной. Например, для функции `CreateRectRgn` группу будут составлять все функции, имеющие отношение к регионам. Если теперь нажимать на кнопку << (два знака "меньше") сразу под главным меню окна справки, то будут появляться страницы с кратким описанием возможных применений объектов, с которыми работают функции (в приведенном примере описание возможностей регионов). Чтобы читать их в нормальной последовательности, лучше всего нажать на кнопку << столько раз, сколько возможно, а затем пойти в противоположном направлении с помощью кнопки >>.

Иногда в справке можно встретить указания "Windows 95 only" или "Windows NT only". К этим замечаниям следует относиться критически, т. к. справка написана для Windows 95, когда еще не было Windows NT 4.0, и описывается версия со старым интерфейсом. Так что то, про что написано "Windows 95 only", может вполне успешно работать и в Windows NT 4.0 и выше, особенно если это "что-то" связано с пользовательским интерфейсом. То же самое относится и к `QuickInfo`. Такие вещи лучше всего проверять на практике.

Еще несколько слов о числовых константах. В справке можно встретить числа вида, например, 0xC56F или 0x3341. Префикс "0x" в Си означает шестнадцатеричное число. В Delphi надо его заменить на "$", т. е. названные числа должны быть записаны как $C56F и $3341 соответственно.

Delphi и функции API

Разработчики Delphi инкапсулировали обширную функциональность Windows API в VCL. Однако WinAPI столь велик и разнообразен, что вряд ли было бы удобно внедрять все функции в объекты Object Pascal. Это, скорее

всего, привело бы к увеличению громоздкости интегрированной среды и соответственно увеличению времени разработки приложений. Поэтому для решения какой-либо специфической задачи разработчик должен уметь использовать арсенал низкоуровневых функций Windows API. Сочетание простоты и естественности использования объектов Delphi со знанием Windows API делает разработчика практически неограниченным в своих замыслах.

Далее авторы постараются кратко ознакомить вас с основными функциями Windows API.

Функции управления окнами

Как следует из названия самой операционной системы Windows, основным элементом для пользователей являются окна. Окна могут быть основными и дочерними, диалоговыми и нет. Но мало кто из начинающих программистов знает, что окнами Windows являются не только видимые на первый взгляд главные окна. Помимо типичных (естественных) окон, окнами в Windows являются также большинство элементов управления, таких как поля ввода, списки, кнопки и т. п.

Фактически любой элемент интерфейса, способный получать фокус ввода, является окном Windows.

Группа функций Windows API, представленных в данном разделе, позволяет программисту напрямую управлять размерами, свойствами и положением окон в операционной системе.

Однако перед тем как мы перейдем непосредственно к рассмотрению данных функций, остановимся на термине "Z-порядок".

Z-порядок

Z-порядок (Z order) окна обозначает позицию окна в стеке перекрывающихся окон. Этот оконный стек ориентируется по воображаемой z-оси, проходящей перпендикулярно поверхности экрана. Окно сверху Z-порядка перекрывает все другие окна. Окно внизу Z-порядка перекрыто всеми другими окнами.

Windows сохраняет Z-порядок в обычном списке. Самое верхнее окно (topmost window) накладывается на все другие окна независимо от того, являются ли они активными или приоритетными окнами. Самое верхнее окно имеет стиль `WS_EX_TOPMOST`. Все самые верхние окна появляются в Z-порядке перед любыми не самыми верхними окнами. Дочернее окно всегда сгруппировано в Z-порядке со своим родителем и другими дочерними окнами этого родителя.

Когда прикладная программа создает окно, Windows помещает его наверху Z-порядка окон, учитывая их родительские отношения. Можно использовать

функцию BringWindowToTop, чтобы перенести окно в верхнюю часть Z-порядка. Можно перестраивать Z-последовательность, используя функции SetWindowPos и DeferWindowPos.

Пользователь изменяет Z-порядок, активизируя различные окна. Windows устанавливает активное окно сверху Z-порядка для окон того же родителя (или поверх всех, если оно не имеет такового). Когда окно переходит в верхнюю часть Z-порядка, его дочерние окна делают это тоже. Можно использовать функцию GetTopWindow, чтобы отыскать все дочерние окна родительского окна и возвратить дескриптор дочернего окна, которое является самым верхним в Z-последовательности. Функция GetNextWindow извлекает данные о дескрипторе следующего или предыдущего окна в Z-последовательности.

Функции движения окном

В табл. 6.2 приведена группа функций API, управляющих движением окна.

Таблица 6.2. *Функции движения окном*

Имя	Описание
AdjustWindowRect	Рассчитывает размеры окна, способного вместить клиентскую область заданного размера
AdjustWindowRectEx	Выполняет те же задачи, что и AdjustWindowRect, но с учетом расширенного стиля окна
BeginDeferWindowPos	Начинает процесс движения нескольких окон одновременно
BringWindowToTop	Активизирует и переносит указанное окно на верхний уровень Z-порядка и накладывает его на другие перекрывающиеся окна
CascadeWindows	Располагает указанные окна каскадом (с наложением)
CloseWindow	Сворачивает указанное окно и отображает его иконку, которая была зарегистрирована с классом окна
DeferWindowPos	Задает новые позицию и размеры окна, которое должно быть перемещено
EndDeferWindowPos	Заканчивает процесс одновременного движения нескольких окон
GetWindowPlacement	Возвращает информацию о видимости и позиции (развернутое, свернутое или нормальное) указанного окна
MoveWindow	Изменяет положение и/или размеры окна
OpenIcon	Восстанавливает и активизирует окно из свернутого

Таблица 6.2 *(окончание)*

Имя	Описание
SetWindowPlacement	Задает состояние (свернутое, развернутое, нормальное) и позицию указанного окна
SetWindowPos	Задает одновременно размеры, позицию и Z-порядок окна. При перемещении окна отличается от DeferWindowsPos тем, что перемещает только одно окно
ShowOwnedPopups	Переключает видимость всех всплывающих окон указанного родительского окна. Это исключает необходимость вызывать функцию ShowWindow для каждого отдельно всплывающего окна
ShowWindow	Показывает, скрывает или изменяет состояние видимости указанного окна на экране. Может также использоваться для свертывания и развертывания окна
ShowWindowAsync	Аналогична функции ShowWindow, кроме того, ставит события показа окна в очередь в многонитевом приложении
TileWindows	Располагает указанные окна рядом друг с другом

Проиллюстрируем работу некоторых функций на следующих далее примерах. Хотелось бы заметить, что в собственной программе можно обойтись и без функций API (например, передвинуть окно в другую позицию), используя свойства объектов Delphi, однако с функциями API эти же действия можно выполнить и над окном чужого приложения (если это окно, конечно, будет способно их выполнить), для этого достаточно только заменить handle формы приложения на текущий handle любого нужного окна.

В листинге 6.1 показан пример, как с помощью функции MoveWindow можно передвинуть некоторое окно (например, панель инструментов) вместе с главным окном.

Функция MoveWindow имеет следующий синтаксис:

```
function MoveWindow(hWnd: HWND; X: Integer; Y: Integer;
        nWidth: Integer;
        nHeight: Integer;
        bRepaint: BOOL ): BOOL;
```

Параметры данной функции представлены в табл. 6.3.

При успешном выполнении функция вернет TRUE, иначе функция возвратит FALSE.

Таблица 6.3. *Параметры* `MoveWindow`

Параметр	Описание
`hWnd`	Дескриптор перемещаемого/изменяемого окна
`X`	Горизонтальная координата, в которую должно быть перемещено окно
`Y`	Вертикальная координата, в которую должно быть перемещено окно
`nWidth`	Новая ширина окна
`nHeight`	Новая высота окна
`bRepaint`	Флаг перерисовки окна: `TRUE`, если функция должна перерисовать окно, `FALSE`, если за обновление окна отвечает приложение

Поскольку в окне свойств объекта `TForm` нет события, возникающего при движении окна, то придется использовать другой способ. При движении любого окна генерируется сообщение `WM_MOVE` *(см. разд. "Функции сообщений окна" данной главы)*, его можно перекрыть собственной процедурой, что и сделано в следующем примере.

Будем перемещать форму `Form2`, поэтому в качестве первого параметра функции указываем дескриптор этой формы. Для перерисовки окна параметр `bRepaint` необходимо установить в `TRUE`, т. к. событие передвижения (а вместе с ним и перерисовка) перекрыты собственным кодом. Исходный код примера можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex01.

В качестве демонстрации использования произвольного handle в каталоге Source\Ch06\Ex02 приведен пример использования `MoveWindow` для перемещения любого выбранного на экране окна.

Листинг 6.1. Перемещение окна с помощью `MoveWindow`

```
type
  TForm1 = class(TForm)
    procedure FormShow(Sender: TObject);
  private
  {нужно перекрыть сообщение WM_MOVE}
  procedure WMMove(var Msg: TWMMove); message WM_MOVE;
  public
    { Public declarations }
  end;
. . . . . .
```

```
procedure TForm1.FormShow(Sender: TObject);
begin
Form2.Show
end;

procedure TForm1.WMMove(var Msg: TWMMove);
begin
{если окно Form2 видимо, то...}
if Form2<>NIL then
{...двигаем Form2 вместе с главной формой}
MoveWindow(Form2.Handle, Form1.Left+Form1.Width+5, Form1.Top,
           Form2.Width, Form2.Height, TRUE);
end;
```

Следующие примеры демонстрируют применение функций `CloseWindow` и `OpenIcon`.

Функции `CloseWindow` и `OpenIcon` имеют следующий сходный синтаксис:

```
CloseWindow(hWnd: HWND): BOOL;
OpenIcon(hWnd: HWND): BOOL;
```

Функции имеют следующие сходные параметры, представленные в табл. 6.4.

Таблица 6.4. *Параметры* `CloseWindow` *и* `OpenIcon`

Параметр	Описание
`hWnd`	Дескриптор окна, к которому применяется действие

При успешном выполнении функции вернут `TRUE`, иначе возвратят `FALSE`.

Пример, представленный в листинге 6.2, демонстрирует работу с перечисленными функциями. Стоит отметить, что пример также использует дескриптор главной формы. Функция API `IsIconic` проверяет, является ли окно свернутым.

Исходный код примера можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\ Ex03.

Для того чтобы применить эти же действия к любому другому окну (находящемуся в момент времени в свернутом или развернутом виде), достаточно передать функциям его дескриптор. Исходный код, представленный на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex04, позволяет использовать данные операции с любым окном Windows.

Листинг 6.2. Сворачивание и разворачивание окна с помощью функций API

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
{если форма свернута, то...}
if IsIconic(Form1.Handle) then
 {...восстанавливаем ее...}
 OpenIcon(Form1.Handle)
else
 {...иначе минимизируем}
 CloseWindow(Form1.Handle);
end;
```

Функции сообщений окна

Окно в Windows имеет массу специфической информации, ассоциированной с ним. Такие детали, как размерность, положение, наследственность и др., могут использоваться или модифицироваться приложением. Windows API имеет набор функций, которые позволяют приложению использовать и изменять практически любую деталь, относящуюся непосредственно к окну или его классу.

Любое окно в системе имеет механизм хранения информации, известный как список свойств. Этот список предназначен исключительно для пользователя и не используется напрямую операционной системой. Каждое окно имеет этот список, включая формы и все элементы управления, порожденные от `TWinControl`.

Список свойств хранится в принадлежащей каждому окну области памяти, которой операционная система управляет автоматически.

Список свойств работает на манер INI-файлов. Функция `SetProp` устанавливает строку и 32-битное целое число. Если строки еще не существует в списке, указанная строка и данные добавляются. Если строка уже существует, то данные этой строки меняются на соответствующее число. Функция `GetProp` извлекает эти свойства, а функция `RemoveProp` удаляет свойство из списка. Приложение не может удалить какое-либо свойство окна, пока оно используется другим приложением, но всегда может удалять любые свойства из списков своих окон.

Списки свойств дают разработчику прекрасную замену использованию глобальных переменных, которые могут иногда вызывать такие ошибки, как выход за пределы использования памяти переменной или совпадение имен. Списки позволяют хранить любое количество информации для любого свойства, предоставляя операционной системе управлять правильным распреде-

лением памяти для этой информации. Для того чтобы вызвать то или иное свойство, достаточно указать ассоциированное с ним число.

В табл. 6.5 приведены функции сообщений окна.

Таблица 6.5. *Функции сообщения окна*

Имя	Описание
`ChildWindowFromPoint`	Определяет дочернее окно, которое содержит заданную точку относительно клиентской области родительского окна
`ChildWindowFromPointEx`	То же, что и `ChildWindowFromPoint`, но при определении дочернего окна позволяет игнорировать запрещенные, прозрачные и невидимые дочерние окна
`EnableWindow`	Разрешает или запрещает окну принимать ввод от мыши или клавиатуры
`EnumChildWindows`	Передает функции обратного вызова дескрипторы дочерних окон, принадлежащих родительскому окну, до тех пор, пока не будут обработаны все дочерние окна или пока функция обратного вызова не вернет `FALSE`
`EnumProps`	Передает функции обратного вызова элементы свойств окна
`EnumPropsEx`	Аналогична функции `EnumProps`, но, кроме того, имеет дополнительный параметр для передачи в функцию обратного вызова данных, определенных в приложении
`EnumThreadWindows`	Передает функции обратного вызова дескриптор каждого окна, принадлежащего нити. Выполняется, пока не будут обработаны все окна в нити или пока функция обратного вызова не вернет `FALSE`
`EnumWindows`	Вызывает функцию обратного вызова, определенную в приложении, для всех окон верхнего уровня, находящихся на экране. Продолжает выполняться, пока все окна верхнего уровня не будут переданы функции обратного вызова или пока функция обратного вызова не возвратит `FALSE`
`FindWindow`	Находит окно верхнего уровня с заданным именем класса и именем окна. Дочерние окна игнорируются
`FindWindowEx`	То же, что и `FindWindow`, но включает в поиск и дочерние окна, которые соответствуют данному имени класса и имени окна

Таблица 6.5 *(продолжение)*

Имя	Описание
FlashWindow	Периодически высвечивает полосу заголовка окна или заставляет мерцать иконку свернутого окна
GetActiveWindow	Возвращает дескриптор активного окна, связанного с нитью, которая вызывает эту функцию
GetClassInfo	Возвращает информацию WNDCLASS для класса окна. Не может возвращать информацию о классе для малой иконки
GetClassInfoEx	Аналогична функции GetClassInfo, но в отличие от нее, выбирает расширенную информацию о классе, в том числе и информацию о классе для малой иконки
GetClassLong	Выбирает 32-разрядное (длинное) значение из структуры WNDCLASS, связанной с классом заданного окна
GetClassName	Возвращает имя класса, с помощью которого было создано заданное окно
GetClientRect	Определяет клиентский прямоугольник для заданного окна, координаты которого выдаются относительно клиентской области окна, и поэтому левый верхний угол всегда имеет координаты (0, 0)
GetDesktopWindow	Возвращает дескриптор окна рабочего стола
GetFocus	Возвращает дескриптор окна, которое имеет текущий фокус ввода
GetForegroundWindow	Возвращает дескриптор окна переднего плана, с которым работает пользователь
GetNextWindow	Возвращает дескриптор следующего или предыдущего по порядку окна. Если заданное окно является дочерним, то следующее или предыдущее также будет дочерним, имеющим то же родительское окно
GetParent	Возвращает дескриптор окна, которое является владельцем заданного окна
GetProp	Возвращает дескриптор элемента свойств, связанный с дескриптором заданного окна
GetTopWindow	Возвращает дескриптор дочернего окна, находящегося на вершине своего Z-порядка
GetWindow	Ищет в списке родительских и дочерних окон то, которое соответствует критериям, заданным во втором параметре
GetWindowLong	Выбирает 32-разрядное (длинное) значение из хранимой информации о заданном окне

Таблица 6.5 *(продолжение)*

Имя	Описание
GetWindowRect	Выбирает информацию о размерах ограничительного прямоугольника окна и копирует ее во второй параметр, имеющий структуру RECT
GetWindowText	Выбирает заголовки родительского, всплывающего и дочернего окон или текст из таких элементов, как кнопка, элемент редактирования и др.
GetWindowTextLength	Применяется для определения длины заголовка окна или текста элемента управления перед вызовом функции GetWindowsText. Резервирует буфер, имеющий достаточный размер для выборки этого текста
IsChild	Проверяет, является ли указанное окно прямым потомком определенного родительского окна
IsIconic	Определяет, находится ли указанное окно в свернутом состоянии
IsWindow	Проверяет, указывает ли дескриптор на допустимое окно
IsWindowEnabled	Определяет состояние окна: разрешено оно или запрещено
IsWindowUnicode	Определяет, предусмотрена ли в заданном окне встроенная поддержка Unicode
IsWindowVisible	Позволяет определить, видимо или не видимо заданное окно на экране (Окна, которые выведены на экран, но затенены другими, считаются видимыми)
IsZoomed	Определяет, развернуто ли окно на весь экран
RemoveProp	Удаляет указанный элемент из списка свойств окна и возвращает дескриптор, связанный с элементом свойств
SetActiveWindow	Делает активным окно верхнего уровня, связанное с вызывающей нитью
SetClassLongPtr	Изменяет одно из значений указателя класса окна
SetFocus	Передает заданному окну фокус ввода
SetForegroundWindow	Активизирует указанное окно и делает его окном переднего плана
SetParent	"Подменяет" родительское окно указанного дочернего окна
SetProp	Добавляет новый элемент свойств в список свойств окна

Таблица 6.5 *(окончание)*

Имя	Описание
SetWindowLongPtr	Изменяет значение указателя, которое связано с окном. Значение должно быть 32-разрядным в 32-разрядной версии Windows и 64-разрядным в 64-разрядной версии
SetWindowText	Изменяет текст, связанный с окном
WindowFromPoint	Находит окно, содержащее заданную точку на экране

Рассмотрим несколько примеров, демонстрирующих работу этих функций.

Попробуем найти все окна верхнего уровня (окна, не являющиеся дочерними по отношению к каким-либо другим окнам). Для перебора окон верхнего уровня используем функцию EnumWidows, а для получения информации об окне — функцию GetWindowText, которая будет вызываться внутри функции обратного вызова.

Функции имеют следующий синтаксис:

Функция EnumWindows:

```
Function EnumWindows(lpEnumFunc: TFNWndEnumProc; lParam: LPARAM ):
                                                              BOOL;
```

имеет следующие параметры, представленные в табл. 6.6.

Таблица 6.6. *Параметры* EnumWindows

Параметр	Описание
lpEnumFunc	Адрес функции обратного вызова, определенной в приложении
lParam	Заданное в приложении 32-разрядное значение, которое будет передано в функцию обратного вызова для каждого окна

При успешном выполнении функция вернет TRUE, иначе — FALSE.

Функция GetWindowText:

```
Function GetWindowText(hWnd: HWND; lpString: PChar;
                       nMaxCount: Integer):  Integer;
```

имеет следующие параметры, представленные в табл. 6.7.

В случае успешного завершения функция возвращает количество символов, скопированных в буфер без учета символа конца строки. В противном случае — 0.

Таблица 6.7. *Параметры* `GetWindowText`

Параметр	Описание
`hWnd`	Дескриптор выбираемого окна
`LpString`	Указатель на буфер, который принимает текст
`nMaxCount`	Максимальное количество символов, вмещающихся в буфер

Программа, перебирающая все окна верхнего уровня, представлена в листинге 6.3. Исходный код примера можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex05.

Листинг 6.3. Перебор всех окон верхнего уровня

```
function EnumerateWindows(hWnd: HWND; lParam: LPARAM): BOOL; stdcall;

procedure TForm1.Button1Click(Sender: TObject);
begin
 {очищаем список}
 ListBox1.Items.Clear;
 {перебираем все окна верхнего уровня в системе}
 EnumWindows(@EnumerateWindows,0); {используется обратный вызов функции
                                    EnumerateWindows}
end;

function EnumerateWindows(hWnd: HWND; lParam: LPARAM): BOOL;
var
 TheText: Array[0..255] of char; {в массив символов записывается текст,
                                  связанный с окном}
begin
 {если окно не содержит описания, то...}
 if (GetWindowText(hWnd, TheText, 255)=0) then
 {...выводим только handle...}
 Form1.ListBox1.Items.Add(Format('%d = {Это окно не имеет
                                     описания}',[hWnd]))
else
 {иначе выводим handle и описание}
 Form1.ListBox1.Items.Add(Format('%d = %s',[hWnd,TheText]));
 {продолжаем перебор}
 Result:=TRUE;
end;
```

На рис. 6.1 показан результат выполнения примера.

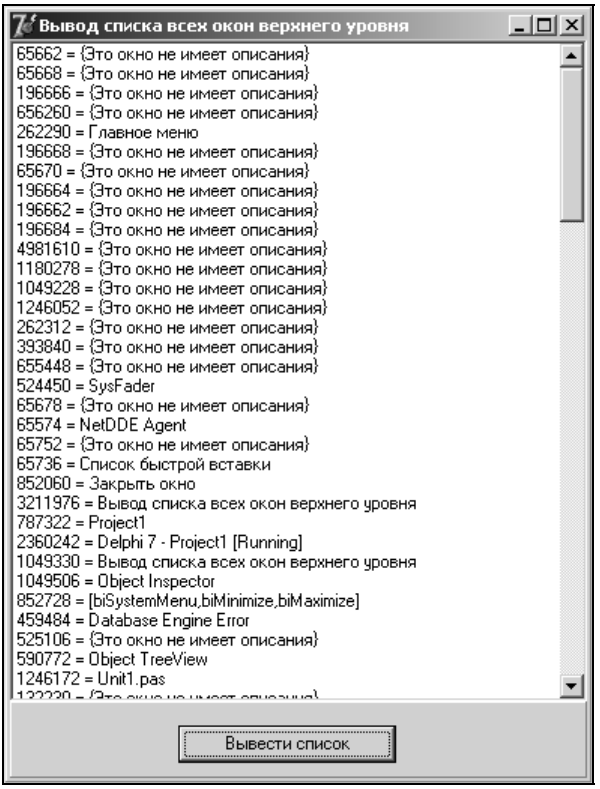


Рис. 6.1. Показ информации об окнах верхнего уровня

В примере, представленном в листинге 6.4, показано, как с помощью функций `WindowFromPoint` и `GetWindowText` можно найти информацию об окне, в область которого попадает курсор мыши. Исходный код примера находится на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex06.

Листинг 6.4. Получение информации об указываемом окне

```
procedure TForm1.Button1Click(Sender: TObject);
var
WindowText: array[0..255] of char; {символьный массив для хранения
                                    текста, связанного с окном}
TheWindow: HWND; {содержит дескриптор окна}
ThePoint: TPoint; {переменная для хранения заданных координат}

begin
{задаем координаты точки экрана}
ThePoint.X := SpinEdit1.Value;
ThePoint.Y := SpinEdit2.Value;
```

```
{вычисляем дескриптор окна формы, в область которого попадают координаты
 заданной точки экрана}
TheWindow := WindowFromPoint(ThePoint);
{получаем текст, связанный с окном}
If (GetWindowText(TheWindow, WindowText, 255))<>0 then
   {...и показываем этот текст}
   Edit1.Text := WindowText
Else
  Edit1.Text := 'Данное окно не имеет описания';

{устанавливаем курсор мыши в заданную точку экрана}
Mouse.CursorPos := ThePoint;
end;
```

Итак, на примерах функций управления окном Win32 API легко понять смысл *дескриптора* ("хендла", *handle*), числового идентификатора любого объекта Windows. Используя дескриптор, вы можете практически "напрямую" с помощью функций API управлять ресурсами операционной системы. Объектом, имеющим свой идентификатор, может быть не только окно, но и, например, файл. В следующем разделе мы разберем работу функций ввода/вывода в файл Windows API.

Функции ввода/вывода в файл

Редкое приложение обходится без работы с файлами, в которых хранятся документы, созданные этим приложением, и различные конфигурационные данные. Стандартного набора объектов Delphi VCL зачастую не хватает для реализации нужной операции с файловой системой. Windows API содержит богатый набор функций для работы с файлами, позволяющий реализовать практически любую задачу.

Любому вновь создаваемому или открываемому на чтение/запись файлу так же, как и любому своему объекту, операционная система присваивает уникальный идентификатор — дескриптор, наподобие дескриптора окна, который служит для обращения к нему. Функция `CreateFile` создает новый или открывает существующий файл и возвращает его дескриптор. Текущую позицию чтения/записи в открытом файле определяет специальный указатель. С помощью функции `SetFilePointer` можно произвольно перемещать указатель в теле файла на заданное число байт. Одному файлу может быть присвоено несколько дескрипторов, если его используют разные приложения.

Пока файл открыт, он управляется с помощью своего дескриптора. Любой дочерний процесс наследует этот дескриптор из родительского процесса. Хотя Windows может закрыть любой открытый файл при завершении приложе-

ния, информация файла будет потеряна, если он был закрыт без использования функции `CloseHandle`.

В стандартном модуле Delphi Sysutils имеются практически все необходимые функции для работы с файлами. Функции API предоставляют более широкие возможности при работе с атрибутами файлов, распределением доступа приложений к ним, отображением файлов.

В табл. 6.8 приведены функции ввода/вывода в файл.

Таблица 6.8. *Функции ввода/вывода в файл*

Имя	Описание
`CloseHandle`	Закрывает открытый дескриптор объекта
`CompareFileTime`	Сравнивает две 64-разрядные типа `TFileTime` отметки времени файла
`CopyFile`	Копирует существующий файл в новый. Атрибуты защиты в новый файл не копируются
`CopyFileEx`	Копирует существующий файл в новый, сохраняя расширенные атрибуты. Атрибуты защиты в новый файл не копируются
`CreateDirectory`	Создает новый каталог
`CreateDirectoryEx`	Аналогична `CreateDirectory` за исключением того, что новый каталог сохраняет атрибуты заданного образца каталога
`CreateFile`	Создает, открывает файл, канал, ресурс связи, дисковое устройство, поток ввода/вывода на консоль, либо меняет их размеры. Может открывать каталог для доступа
`CreateFileMapping`	Создает именованный или неименованный объект отображения файла
`DeleteFile`	Удаляет указанный существующий файл
`DosDateTimeToFileTime`	Преобразует значения даты и времени MS DOS в 64-разрядную отметку времени
`FileTimeToDosDateTime`	Конвертирует 64-разрядную отметку времени в MS DOS-формат
`FileTimeToLocalFileTime`	Преобразует время из формата UTC (Universal Coordinated Time — универсальное скоординированное время) в местное время
`FileTimeToSystemTime`	Конвертирует 64-разрядную отметку времени в системный формат времени типа `TSystemTime`

Таблица 6.8 *(продолжение)*

Имя	Описание
FindClose	Закрывает дескриптор поиска, который был создан ранее с помощью FindFirstFile и применялся в функции FindNextFile
FindCloseChangeNotification	Прекращает отслеживание извещений дескриптора контроля изменений
FindFirstChangeNotification	Создает дескриптор контроля изменений и устанавливает начальные условия фильтрации извещений об изменениях
FindFirstFile	Ищет в каталоге первый файл, имя которого соответствует указанному искомому имени
FindNextChangeNotification	Запрашивает следующий дескриптор контроля изменений
FindNextFile	Ищет следующий файл с указанным именем. Может использоваться только после вызова FindFirstFile
FlushFileBuffers	Вызывает запись всех буферизированных данных в файл и очищает буферы файла
FlushViewOfFile	Переписывает на диск байты из отображенного в память представления файла
GetCurrentDirectory	Возвращает полный путь текущего каталога
GetFileAttributes	Возвращает атрибуты файла или каталога
GetFileAttributesEx	Возвращает расширенные атрибуты файла или каталога
GetFileInformationByHandle	Позволяет получить информацию о файле, который обозначен дескриптором файла
GetFileSize	Определяет размер файла в байтах
GetFileTime	Определяет дату и время создания, последнего обращения и последнего изменения файла
GetFileType	Возвращает тип файла с указанным дескриптором
GetFullPathName	Определяет имя заданного файла и полный путь к нему
GetShortPathName	Возвращает короткий путь (в формате 8.3) для указанного файла
GetTempFileName	Создает временное имя файла
GetTempPath	Определяет путь к каталогу, предназначенному для временных файлов

Таблица 6.8 *(окончание)*

Имя	Описание
LocalFileTimeToFileTime	Преобразует отметку местного времени в отметку времени в формате UTC (Universal Coordinate Time — универсальное скоординированное время)
LockFile	Блокирует участок в открытом файле и предоставляет блокирующему процессу монопольный доступ к указанному участку
MapViewOfFile	Отображает указанный файл в адресное пространство вызывающего процесса
MoveFile	Перемещает (переименовывает) существующий файл или каталог в новое место на томе
OpenFileMapping	Открывает именованный объект отображения файла
ReadFile	Считывает данные из файла, начиная с позиции, обозначенной указателем файла
RemoveDirectory	Удаляет указанный пустой каталог
SearchPath	Выполняет поиск указанного файла
SetCurrentDirectory	Делает текущим указанный каталог
SetEndOfFile	Перемещает позицию конца файла (EOF) в текущую позицию указателя файла
SetFileAttributes	Устанавливает (изменяет) атрибуты файла
SetFilePointer	Перемещает указатель открытого файла
SetFileTime	Устанавливает (изменяет) дату и время создания, последнего обращения или последнего изменения файла
SystemTimeToFileTime	Преобразует значение системного времени в отметку времени в формате UTC (Universal Coordinate Time — универсальное скоординированное время)
UnlockFile	Разблокирует заблокированный участок в открытом файле
UnmapViewOfFile	Отменяет отображение представления файла в адресном пространстве вызывающего процесса
WriteFile	Выполняет запись данных в файл, начиная с позиции указателя файла

Рассмотрим основные функции ввода/вывода в файл. Прежде всего рассмотрим функцию CreateFile, позволяющую создавать или открывать файл для чтения/записи в него.

Итак, функция CreateFile создает новый или открывает существующий файл и имеет вид:

```
Function CreateFile(lpFileName: PChar;
            dwDesiredAccess: Integer;
            dwShareMode: Integer;
            lpSecurityAttributes: PSecurityAttributes;
            dwCreationDisposition: DWORD;
            dwFlagsAndAttributes: DWORD;
            hTemplateFile: THandle; ): THandle;
```

Параметры функции представлены в табл. 6.9.

Таблица 6.9. *Параметры функции* `CreateFile`

Параметр	Описание
lpFileName	Содержит путь и имя создаваемого или открываемого файла (канала, ресурса связи, дискового устройства, консоли или другого объекта)
dwDesiredAccess	Содержит флаги типа доступа (на чтение и/или запись)
dwShareMode	Содержит флаги, определяющие способ установки совместного доступа
lpSecurityAttributes	Указатель на структуру TSecurityAttributes, содержащую атрибуты безопасности файла (для NTFS)
dwCreationDisposition	Содержит один из флагов, определяющих действие, которое необходимо выполнить в случае, если файл существует, и в случае, когда он не существует
dwFlagsAndAttributes	Содержит флаги атрибутов и доступа, возможна их любая комбинация, однако любые атрибуты отменяют действие FILE_ATTRIBUTE_NORMAL
hTemplateFile	Дескриптор с правами доступа GENERIC_READ к шаблону файла

В случае успешного выполнения функция вернет дескриптор открытого файла. В табл. 6.10 представлены расширенные описания значений параметров функции CreateFile.

Если файл существовал до вызова функции и параметр dwCreate имеет значение CREATE_ALWAYS или OPEN_ALWAYS, то вызов функции GetLastError вернет

сообщение об ошибке ERROR_ALREADY_EXISTS, а если файл не существовал, то GetLastError вернет 0. Если функция завершается неуспешно, то она возвращает дескриптор со значением INVALID_HANDLE_VALUE.

Таблица 6.10. *Значения параметров функции* `CreateFile`

Значение	Описание
dwDesiredAccess	
0	Разрешить приложению запрашивать атрибуты устройства (без доступа нему)
GENERIC_READ	Определяет доступ для чтения из файла
GENERIC_WRITE	Определяет доступ для записи в файл
DwShareMode	
0	Запретить совместное использование файла
FILE_SHARE_DELETE	(Только для платформ NT.) Открытие файла будет успешным, только если затребован доступ для удаления
FILE_SHARE_READ	Файл может открываться для общего доступа на чтение
FILE_SHARE_WRITE	Файл может открываться для общего доступа на запись
dwCreationDisposition	
CREATE_NEW	Создать новый файл. Функция завершается неудачно, если файл уже существует
CREATE_ALWAYS	Создать новый файл. Если файл уже существует, то он будет перезаписан
OPEN_EXISTING	Открыть существующий файл. Если файл не существует, то функция завершится неудачно
OPEN_ALWAYS	Открыть файл. Если он не существует, то будет создан
TRUNCATE_EXISTING	Открыть файл и установить его размер в 0 байт. Вызывающий процесс должен открыть файл с доступом не ниже GENERIC_WRITE
dwFlagsAndAttributes	
FILE_ATTRIBUTE_ARCHIVE	Файл является архивным. Его можно пометить для операций резервного копирования или удаления
FILE_ATTRIBUTE_ATOMIC	Зарезервированное значение; не должно использоваться

Таблица 6.10 *(продолжение)*

Значение	Описание
FILE_ATTRIBUTE_COMPRESSED	Файл или каталог сжат. Если сжат каталог, то создаваемые в нем файлы или подкаталоги будут сжаты по умолчанию
FILE_ATTRIBUTE_DIRECTORY	Файл является каталогом
FILE_ATTRIBUTE_ENCRYPTED	(Толька для платформ NT.) Файл или каталог зашифрован. Если зашифрован каталог, то для вновь создаваемых в нем файлов будет применяться шифрование по умолчанию. Действие этого флага отменяется, если установлен флаг FILE_ATTRIBUTE_SYSTEM
FILE_ATTRIBUTE_HIDDEN	Файл является скрытым
FILE_ATTRIBUTE_NORMAL	Файл не имеет атрибутов
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	(Только для платформ NT.) Файл не должен быть проиндексирован службой индексации
FILE_ATTRIBUTE_OFFLINE	Данные файла физически перемещены на автономное запоминающее устройство и недоступны непосредственно
FILE_ATTRIBUTE_READONLY	Файл предназначен только для чтения
FILE_ATTRIBUTE_SYSTEM	Файл является системным
FILE_ATTRIBUTE_TEMPORARY	Файл является временным
FILE_ATTRIBUTE_XACTION_WRITE	Зарезервированное значение; не должно использоваться
FILE_FLAG_BACKUP_SEMANTICS	(Только для платформ NT.) Открывает или создает файл для операции резервного копирования или восстановления
FILE_FLAG_ DELETE_ON_CLOSE	Файл должен быть удален сразу после того, как будут удалены все его дескрипторы
FILE_FLAG_NO_BUFFERING	Файл открыт без промежуточной буферизации или кэширования. Это позволяет в некоторых случаях повысить производительность. Приложение при этом должно отвечать следующим требованиям: • доступ к файлу должен начинаться со смещений в файле, кратных размеру сектора тома; • ввод/вывод должен быть кратен сектору тома;

Таблица 6.10 *(продолжение)*

Значение	Описание
	• адреса буферов для операций чтения/записи должны быть выровнены по границам адресов памяти, кратных размеру сектора тома. В приложении можно определить размер сектора тома, используя функцию `GetDiskFreeSpace`
`FILE_FLAG_OPEN_REPARSE_POINT`	(Только для платформ NT.) Запретить применять обычные правила в точках повторной интерпретации NTFS. При открытии файла возвращается дескриптор файла независимо от того, действует ли фильтр, управляющий точкой повторной интерпретации
`FILE_FLAG_OPEN_NO_RECALL`	Указывает, что затребованы данные файла, но файл должен по-прежнему находиться на удаленном запоминающем устройстве. Флаг предназначен для использования в системах управления удаленными запоминающими устройствами или в системах иерархического управления памятью
`FILE_FLAG_OVERLAPPED`	Выполнять перекрывающиеся операции чтения и записи. Операции `ReadFile`, `WriteFile`, `ConnectedNamedPipe` и `TransactNamedPipe`, для выполнения которых требуется много времени, возвращают сообщения об ошибке `ERROR_IO_PENDING`. Если установлен этот флаг, то в качестве одного из параметров должна быть указана структура `OVERLAPPED`, которой передается указатель позиции в файле
`FILE_FLAG_POSIX_SEMANTICS`	Указывает, что доступ к файлу должен осуществляться согласно правилам POSIX
`FILE_FLAG_RANDOM_ACCESS`	Оптимизировать кэширование для произвольного доступа к файлу
`FILE_FLAG_SEQUENTIAL_SCAN`	Оптимизировать кэширование для последовательного доступа к файлу
`FILE_FLAG_WRITE_THROUGH`	Запись будет выполняться непосредственно в файл, обходя кэш
`hTemplateFile`	
`SECURITY_ANONYMOUS`	Сервер может выступать в роли клиента в режиме анонимного доступа
`SECURITY_CONTEXT_TRACKING`	Указывает, что режим контроля защиты является динамическим. Если флажок не указан, режим контроля защиты является статическим

Таблица 6.10 *(окончание)*

Значение	Описание
SECURITY_DELEGATION	Сервер может выступать в роли клиента в режиме делегирования
SECURITY_EFFECTIVE_ONLY	Указывает, что серверу доступны только определенные свойства контекста защиты клиента. Если этот флажок не указан, серверу доступны все свойства контекста защиты клиента
SECURITY_IDENTIFICATION	Сервер может выступать в роли клиента в режиме идентификации
SECURITY_IMPERSONATION	Сервер может выступать в роли клиента в режиме Impersonation

Непосредственно чтение и запись данных производится функциями `ReadFile` и `WriteFile`, имеющими следующий вид:

Функция `ReadFile`:

```
Function ReadFile(hFile: THandle; var Buffer;
                  nNumberOfBytesToRead: DWORD;
                  var lpNumberOfBytesRead: LPDWORD;
                  lpOverlapped: POverLapped): BOOL;
```

Параметры функции `ReadFile` представлены в табл. 6.11.

Таблица 6.11. *Параметры функции* `ReadFile`

Параметр	Описание
hFile	Дескриптор открытого файла
Buffer	Указатель на буфер, который принимает данные, считанные из файла
nNumberOfBytesToRead	Число байтов, которое должно быть считано из файла
lpNumberOfBytesRead	Указатель на переменную `DWORD`, которая принимает число считанных байтов
lpOverlapped	Указатель на структуру `OVERLAPPED`. Эта структура требуется, если файл, обозначенный дескриптором `hFile`, был создан с флагом `FILE_FLAG_OVERLAPPED`

Функция в случае успешного выполнения вернет `TRUE`, в противном — `FALSE`.

После считывания данных указатель файла сдвигается на число считанных байт при условии, что дескриптор файла не был создан с атрибутом для перекрывающегося ввода/вывода (FILE_FLAG_OVERLAPPED). Если дескриптор файла был создан для перекрывающегося ввода/вывода, приложение должно с помощью функции SetFilePointer корректировать положение после считывания.

Функция WriteFile:

```
Function WriteFile(hFile: THandle; const Buffer;
                   nNumberOfBytesToWrite: DWORD;
                    var lpNumberOfBytesWritten: DWORD;
                    lpOverlapped: POverlapped): BOOL;
```

Параметры функции WriteFile представлены в табл. 6.12.

Таблица 6.12. *Параметры функции* `WriteFile`

Параметр	Описание
hFile	Дескриптор открытого файла
Buffer	Указатель на буфер, содержащий данные, которые должны быть записаны в файл
nNumberOfBytesToWrite	Число байтов, предназначенных для записи в файл
lpNumberOfBytesWritten	Указатель на переменную, которая принимает реально записанное число байтов, записанных в текущем вызове функции. Перед записью или проверкой ошибок функция устанавливает это значение равным нулю
lpOverlapped	Указатель на структуру OVERLAPPED. Этот параметр не должен быть равен NULL, если файл, обозначенный дескриптором hFile, был создан с флагом FILE_FLAG_OVERLAPPED, при этом запись в файл начнется со смещения, указанного в структуре OVERLAPPED, и функция не выполнит возврат до тех пор, пока не будет закончена операция записи

Функция в случае успешного выполнения вернет TRUE, в противном случае — FALSE.

Функция начинает запись данных с позиции, обозначенной указателем файла. После выполнения записи сдвигается на число фактически записанных байтов при условии, что дескриптор файла не был создан с атрибутом для перекрывающегося ввода/вывода (FILE_FLAG_OVERLAPPED). Если дескриптор файла

был создан для перекрывающегося ввода/вывода, приложение должно с помощью функции SetFilePointer корректировать положение после записи.

Функция FlushFileBuffers вызывает запись всех буферизированных данных в файл и очищает буферы файла. Функция имеет вид:

```
Function FlushFileBuffers(hFile:THandle): BOOL;
```

Параметры функции FlushFileBuffers представлены в табл. 6.13.

Таблица 6.13. *Параметры функции* `FlushFileBuffers`

Параметр	Описание
hFile	Дескриптор открытого файла

Функция в случае успешного выполнения вернет TRUE, в противном случае — FALSE.

Прежде чем идти дальше, мы должны изучить возможности управления блокировками файла на уровне Windows API, т. к. знание этих функций имеет большое значение в понимании работы механизма ввода/вывода.

Функция LockFile блокирует файл (или участок файла) для невозможности использования его другими процессами, а UnlockFile разблокирует его.

Функция LockFile имеет вид:

```
Function LockFile(
                hFile: THandle;
                dwFileOffsetLow: DWORD;
                dwFileOffsetHigh: DWORD;
                nNumberOfBytesToLockLow: DWORD;
                nNumberOfBytesToLockHigh: DWORD): BOOL;
```

Параметры функции LockFile представлены в табл. 6.14.

Таблица 6.14. *Параметры функции* `LockFile`

Параметр	Описание
hFile	Дескриптор открытого файла
dwFileOffsetLow	Младшие 32 бита начального байтового смещения в файле, с которого должна начаться блокировка
dwFileOffsetHigh	Старшие 32 бита начального байтового смещения в файле, с которого должна начаться блокировка
nNumberOfBytesToLockLow	Младшие 32 бита значения длины блокируемой области в байтах

Таблица 6.14 *(окончание)*

Параметр	Описание
nNumberOfBytesToLockHigh	Старшие 32 бита значения длины блокируемой области в байтах

Функция в случае успешного выполнения вернет TRUE, в противном случае — FALSE.

Функция разблокировки UnlockFile имеет вид:

```
Function UnlockFile(
                hFile: THandle;
                dwFileOffsetLow: DWORD;
                dwFileOffsetHigh: DWORD;
                nNumberOfBytesToUnlockLow: DWORD;
                nNumberOfBytesToUnlockHigh: DWORD): BOOL;
```

Параметры функции UnlockFile представлены в табл. 6.15.

Таблица 6.15. *Параметры функции* UnlockFile

Параметр	Описание
hFile	Дескриптор открытого файла
dwFileOffsetLow	Младшие 32 бита начального байтового смещения в файле, с которого начинается заблокированная запись
dwFileOffsetHigh	Старшие 32 бита начального байтового смещения в файле, с которого начинается заблокированная запись
nNumberOfBytes ToUnlockLow	Младшие 32 бита длины байтовой области, которая должна быть разблокирована
nNumberOfBytes ToUnlockHigh	Старшие 32 бита длины байтовой области, которая должна быть разблокирована

Как и для LockFile, функция вернет TRUE в случае успешного выполнения, в противном случае — FALSE.

После того как мы рассмотрели основные функции ввода/вывода, еще раз напомним, что по окончании всех операций с файлом необходимо закрыть его дескриптор функцией CloseHandle. Данная функция не только закроет файл, но и произведет корректное освобождение дескриптора для последующего его использования. Иначе к данному дескриптору нельзя будет обратиться либо получить монопольный доступ даже из приложения, которое вызывало его в последний раз.

Итак, функция `CloseHandle` объявляет недействительным дескриптор объекта, уменьшает на единицу число дескрипторов объекта и проверяет сохранность объекта. После закрытия последнего дескриптора объект удаляется из памяти.

> ***Примечание***
>
> Функция также закрывает дескрипторы объектов консоли, файла события, отображения файла, взаимоисключающей блокировки, именованного канала, процесса, семафора, нити, лексемы (в Windows 2000 и старше).

Для демонстрации работы основных функций ввода/вывода мы составили небольшую программу, исходный код которой находится на прилагаемом компакт-диске в каталоге Source\Ch06\Ex07.

Сокращенный исходный код представлен в листинге 6.5. На рис. 6.2 представлен результат работы программы.

Листинг 6.5. Чтение/запись данных в файл

```
{структура записи для хранения данных, введенных в форму}
type
  Information = record
    Name: array[0..255] of char;
    Title: array[0..255] of char;
    Age: Integer;
  end;

procedure TForm1.Button1Click(Sender: TObject);
var
 FileHandle: THandle; // Дескриптор открываемого файла
 TheInfo: Information; // Переменная, содержащая данные типа Information,
 NumBytesWritten: DWORD; // содержит записанное количество байт
begin
 {копируем данные из формы в переменную типа Information}
 StrPCopy(TheInfo.Name, Edit1.Text);
 StrPCopy(TheInfo.Title, Edit2.Text);
 TheInfo.Age := StrToInt(Edit3.Text);
 {создаем бинарный файл}
 FileHandle := CreateFile(Pchar(Edit4.Text), GENERIC_WRITE, 0, nil,
                          CREATE_ALWAYS, FILE_ATTRIBUTE_ARCHIVE, 0);
 {записываем данные, содержащиеся в переменной TheInfo, прямо в файл}
 WriteFile(FileHandle, TheInfo, SizeOf(Information),
           NumBytesWritten, nil);
 Label5.Caption := 'Записано байт: ' + IntToStr(NumBytesWritten);
 {скидываем все буферизированные для записи данные на диск}
 FlushFileBuffers(FileHandle);
```

```
 {закрываем файл}
 CloseHandle(FileHandle);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
 FileHandle: THandle; // Дескриптор открываемого файла
 TheInfo: Information;// Переменная, содержащая данные типа Information,
 NumBytesRead: DWORD; // для запоминания реально считанного количества
                      // байт
 TheTitle: array[0..255] of char; // Для данных, считанных из середины
                                  // файла
begin
 {открываем существующий файл для чтения}
 FileHandle := CreateFile(Pchar(Edit4.Text), GENERIC_READ, 0, nil,
                          OPEN_Existing, FILE_ATTRIBUTE_NORMAL or
                          FILE_FLAG_SEQUENTIAL_SCAN, 0);
 {проверяем корректность дескриптора, если он некорректный, то файл,
  скорее всего, не существует}
 if FileHandle=INVALID_HANDLE_VALUE then
 begin
    ShowMessage('Файл еще не существует. Нажмите кнопку ''Записать в' +
                'файл'', ' +
                'чтобы создать его.');
    Exit;
  end;
 {блокируем файл, чтобы другой процесс не мог его считать}
 LockFile(FileHandle, 0, 0, SizeOf(Information), 0);
 {считываем блок данных в нашу структуру TheInfo}
 ReadFile(FileHandle, TheInfo, SizeOf(Information), NumBytesRead, nil);
 {выводим данные в форму}
 Edit1.Text := TheInfo.Name;
 Edit2.Text := TheInfo.Title;
 Edit3.Text := IntToStr(TheInfo.Age);
 Label5.Caption := 'Считано байт: ' + IntToStr(NumBytesRead);
 {разблокируем и закрываем файл}
 UnlockFile(FileHandle, 0, 0, SizeOf(Information), 0);
 CloseHandle(FileHandle);
end;
```

Следующий пример позволяет редактировать атрибут времени последнего изменения файла. Сокращенный исходный код примера приведен в листинге 6.6.

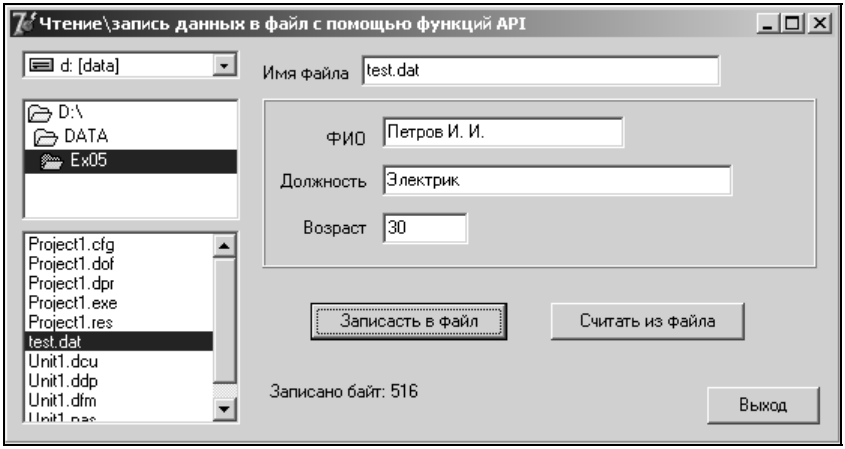


Рис. 6.2. Демонстрация функций чтения/записи

Полный исходный код представлен на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex08. Результат работы программы показан на рис. 6.3.

Листинг 6.6. Редактирование времени последнего изменения файла

```
procedure TForm1.ButtonOpenClick(Sender: TObject);
var
  Security: TSecurityAttributes; {переменная для хранения атрибутов
открываемого файла}
  FileName: PChar; {переменная для хранения имени файла}
  WriteTime, LocalTime: TFILETIME; {переменная для хранения времени
файла}
  DosDate, DosTime: WORD; {переменные для времени и даты в DOS-формате}
  infoDosTime: TDosTime; {переменная для хранения времени в DOS-формате}
  infoDosDate: TDosDate; {переменная для хранения даты в DOS-формате}
  SystemTime: TSystemTime; {переменная для хранения времени последнего
изменения файла}
begin
 {устанавливаем атрибуты безопасности открываемого файла}
 Security.nLength := SizeOf(TSecurityAttributes);
 Security.lpSecurityDescriptor := nil;
 Security.bInheritHandle := FALSE;
 {отображаем диалог открытия файла}
 if OpenDialog1.Execute then
 begin
  {загружаем имя файла...}
  FileName := PChar(OpenDialog1.FileName);
  StatusBar1.SimpleText := FileName;
```

```
  {...и открываем его, получая его дескриптор}
  hFile := CreateFile(PChar(FileName),GENERIC_READ or GENERIC_WRITE,
  FILE_SHARE_READ or FILE_SHARE_WRITE, @Security,
  OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
  {в случае ошибки выводим сообщение}
  if hFile = INVALID_HANDLE_VALUE then
  begin
    ShowMessage('Ошибка открытия файла');
    Exit;
  end;
 end;
 {получаем время его последнего изменения}
 GetFileTime(hFile, nil, nil, @WriteTime);
 {конвертируем время в формат времени локального файла}
 FileTimeToLocalFileTime(WriteTime, LocalTime);
 {конвертируем локальное время в системное время}
 FileTimeToSystemTime(LocalTime, SystemTime);
 {конвертируем время в DOS-формат...}
 FileTimeToDosDateTime(LocalTime, DosDate, DosTime);
 {...и обратно}
 if not DosDateTimeToFileTime(DosDate, DosTime, LocalTime) then
    ShowMessage ('Ошибка конвертирования в DOS-формат и обратно');
 {переводим DOS-время в системное время и
 DOS-дату в системную дату}
 infoDosTime := ConvertDosTimeToSystemTime(DosTime);
 infoDosDate := ConvertDosDateToSystemDate(DosDate);
 Edit1.Text  := IntToStr(infoDosDate.Day) + '.' +
                IntToStr(infoDosDate.Month) + '.' +
                IntToStr(infoDosDate.Year) + ' ' +
         IntToStr(infoDosTime.Hour) + ':' +
         IntToStr(infoDosTime.Minutes) + ':' +
         IntToStr(infoDosTime.Seconds);
 {отображаем дату последнего изменения файла}
 SpinEdit1.Value := SystemTime.wYear;
 SpinEdit2.Value := SystemTime.wHour;
 SpinEdit3.Value := SystemTime.wMinute;
 SpinEdit4.Value := SystemTime.wSecond;
 ComboBox1.ItemIndex := SystemTime.wMonth - 1;
 Calendar1.Month := SystemTime.wMonth;
 Calendar1.Day := SystemTime.wDay;
end;

procedure TForm1.ButtonApplyClick(Sender: TObject);
var
 FileTime, LocalFileTime: TFileTime; {хранит время файла}
 SystemTime: TSystemTime; {хранит системное время}
```

```
begin
 {заполняем поля системного времени из значений, заданных пользователем}
 SystemTime.wHour := SpinEdit2.Value;
 SystemTime.wMinute := SpinEdit3.Value;
 SystemTime.wSecond := SpinEdit4.Value;
 SystemTime.wYear := SpinEdit1.Value;
 SystemTime.wMonth := ComboBox1.ItemIndex + 1;
 SystemTime.wDay := Calendar1.Day;
 {перевод системного времени в формат времени файла}
 SystemTimeToFileTime(SystemTime, LocalFileTime);
 {конвертируем формат времени локального файла в формат, понимаемый
 файловой системой}
 LocalFileTimeToFileTime(LocalFileTime, FileTime);
 {устанавливаем для выбранного файла новое время последнего изменения,
 которое будут отображать файловые менеджеры}
 SetFileTime(hFile, nil, nil, @FileTime);
 {отображаем новые установленные атрибуты времени файла}
 Edit1.Text := IntToStr(Calendar1.Day) + '.' +
               IntToStr(ComboBox1.ItemIndex + 1) + '.' +
               IntToStr(SpinEdit1.Value) + ' ' +
               IntToStr(SpinEdit2.Value) + ':' +
               IntToStr(SpinEdit3.Value) + ':' +
               IntToStr(SpinEdit4.Value);
end;
```

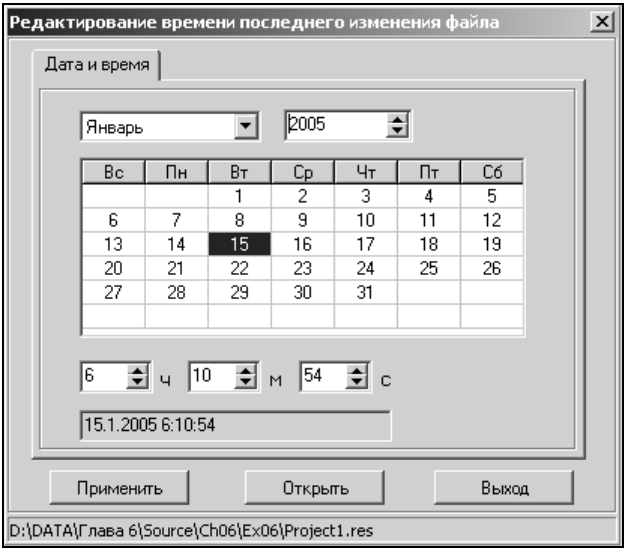


Рис. 6.3. Изменение времени создания файла

Функции ввода

Windows обеспечивает передачу ввода приложению со множества различных устройств, таких как клавиатура, мышь, джойстик и др. Ресурсы ввода могут использоваться совместно различными приложениями.

Windows достаточно корректно управляет вводом в различные приложения в мультизадачной среде. Тем не менее приложение может иметь собственный механизм управления и мониторинга систем ввода. Функции Windows API позволяют обеспечить приложение практически любым нужным механизмом ввода. Наиболее употребительными средствами ввода являются клавиатура и мышь.

Windows обеспечивает работу клавиатуры в контексте интернациональных кодировок. Клавиатура находится в состоянии постоянного опроса и отслеживает любое нажатие клавиши или комбинации клавиш пользователем. Клавиши имеют так называемые виртуальные коды, не зависящие от языка и позволяющие программисту создавать интернациональные приложения, поддерживающие несколько языков ввода. Операционная система производит большую часть низкоуровневой работы по трансляции кодов в символы текущего языка. Каждой национальной раскладке соответствует свой локальный идентификатор, который включает в себя информацию о своем языке и о текущей физической раскладке клавиш. Локальный идентификатор языка ввода может меняться динамически один на другой в зависимости от выбора пользователя, при этом генерируется сообщение `WM_INPUTLANGCHANGE`. События от клавиатуры могут быть эмулированы с помощью функции `keybd_event`. Это сообщение, которое Windows генерирует при реальном нажатии клавиши. Сообщение клавиатуры обычно посылается окну, имеющему фокус ввода.

Мышь является другим разделяемым устройством ввода, состояние которого, аналогичное клавиатуре, постоянно опрашивается операционной системой. При перемещении мыши операционной системой генерируются сообщения, которые могут быть перехвачены приложениями. Анализируя данные сообщения, можно определить, что, например, находится под курсором мыши в конкретный момент.

С помощью функции `ClipCursor` можно ограничить движение курсора мыши какой-либо прямоугольной областью. Аналогично `keybd_event mouse_event` генерирует события мыши, что также позволяет программно эмулировать ее. Эмуляция клавиатуры и мыши позволяет создавать демонстрационные программы, а также посылать сообщения интерфейсу другого приложения.

В табл. 6.16 представлены основные функции ввода.

Таблица 6.16. *Функции ввода*

Имя	Описание
ActivateKeyboardLayout	Активизирует другую раскладку клавиатуры
ClipCursor	Ограничивает перемещение курсора прямоугольной областью на экране
CreateCaret	Создает новую форму курсора каретки
CreateCursor	Создает новую форму курсора мыши
DragDetect	Перехватывает события мыши и следит за ее перемещением до тех пор, пока пользователь не отпустит левую кнопку мыши, не нажмет <Esc> или не переместит курсор мыши за пределы прямоугольника перетаскивания, расположенного вокруг указанной точки
GetAsyncKeyState	Определяет, нажата ли в настоящее время какая-либо клавиша и была ли она нажата после последнего вызова функции GetAsyncKeyState
GetCapture	Возвращает дескриптор окна, которое перехватывает события мыши (если таковое окно имеется)
GetCaretBlinkTime	Возвращает время в миллисекундах между периодическим высвечиванием курсора каретки
GetCaretPos	Находит позицию курсора каретки в клиентских координатах
GetClipCursor	Находит экранные координаты прямоугольной области, которой в последний раз был ограничен курсор мыши с помощью функции ClipCursor
GetCursor	Возвращает дескриптор текущего курсора мыши
GetCursorPos	Находит позицию курсора в экранных координатах
GetDoubleClickTime	Определяет текущий заданный интервал между одиночными щелчками для двойного щелчка
GetInputState	Определяет, есть ли в очереди сообщений вызывающей нити сообщения кнопок мыши или клавиатуры
GetKeyboardLayout	Выбирает активную раскладку для указанной нити
GetKeyboardLayoutList	Находит дескрипторы раскладок клавиатуры, загруженных в настоящее время в системе
GetKeyboardLayoutName	Находит имя текущей раскладки клавиатуры, вызывающей нити
GetKeyboardState	Копирует состояние всех 256 виртуальных клавиш в указанный буфер
GetKeyboardType	Возвращает информацию о текущей клавиатуре

Таблица 6.16 *(продолжение)*

Имя	Описание
GetKeyNameText	Выбирает строку, содержащую имя клавиши
GetKeyState	Возвращает состояние указанной виртуальной клавиатуры
HideCaret	Делает курсор каретки невидимым
LoadCursor	Загружает указанный ресурс курсора из файла выполняемой программы или DLL, связанной с текущим дескриптором экземпляра
LoadKeyboardLayout	Загружает новую раскладку клавиатуры в систему
MapVirtualKey	Преобразует коды виртуальных клавиш
MapVirtualKeyEx	Преобразует коды виртуальных клавиш с использованием входного языка и физической раскладки клавиатуры
OemKeyScan	Конвертирует коды ASCII OEM от 0 до 0xFF в сканкоды OEM и состояния переключения регистра
ReleaseCapture	Останавливает перехват событий мыши в окне текущей нити и восстанавливает нормальную обработку сообщений мыши
SetCapture	Устанавливает перехват событий мыши для указанного окна, принадлежащего к текущей нити, чтобы сообщения мыши принимала только программа с перехваченной мышью
SetCaretBlinkTime	Устанавливает (изменяет) периодичность мерцания курсора каретки в миллисекундах
SetCaretPos	Устанавливает (переводит) курсор каретки в указанные координаты
SetCursor	Позволяет изменить форму курсора
SetCursorPos	Устанавливает (переводит) курсор мыши в указанные экранные координаты
SetDoubleClickTime	Устанавливает (изменяет) интервал между щелчками в миллисекундах для двойного щелчка мыши
SetKeyboardState	Копирует 256-байтный массив состояний клавиш в таблицу состояний ввода с клавиатуры вызывающей нити
ShowCaret	Делает курсор каретки видимым на экране в текущей позиции вставки
ShowCursor	Отображает или скрывает курсор мыши на экране

Таблица 6.16 *(окончание)*

Имя	Описание
SwapMouseButton	Изменяет на противоположные или восстанавливает назначения левой и правой кнопок мыши. Действует только в текущем сеансе Windows
UnloadKeyboardLayout	Удаляет указанную раскладку клавиатуры
VkKeyScan	Преобразует символ в соответствующий код виртуальной клавиши и состояние переключения регистра
VkKeyScanEx	Аналогична функции VkKeyScan за исключением того, что преобразует символы с использованием входного языка и физической раскладки клавиатуры

Рассмотрим подробнее использование некоторых функций на примерах.

Пример, полный исходный код которого находится на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex09, демонстрирует получение информации об установленной в системе клавиатуре. Результат выполнения данного примера представлен на рис. 6.4.

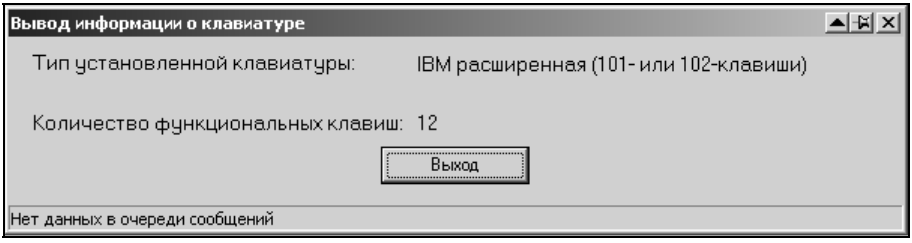


Рис. 6.4. Получение информации о клавиатуре

Следующий пример, сокращенный код которого представлен в листинге 6.7, демонстрирует возможность Windows API по управлению клавиатурными раскладками. В данном примере можно устанавливать, удалять и активизировать языковые раскладки клавиатуры. В примере использованы следующие API-функции: LoadKeyboardLayout, UnloadKeyboardLayout, GetKeyboardLayoutList, GetKeyboardLayoutName и ActivateKeyboardLayout.

Исходный код данного примера находится на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex10.

Листинг 6.7. Управление языковыми раскладками клавиатуры

```
var
  Reg: Tregistry;
  L: TStringList;  // Массив для хранения ключей реестра, содержащих
идентификаторы раскладок
```

```
  List : array [0..MAX_HKL] of HKL; // Массив дескрипторов клавиатуры
  pwszKLID: PChar; // для хранения имени раскладки,
  MyListIndex: Integer; // для хранения индекса раскладки
  LangID: PChar; // Идентификатор раскладки, выбираемый из списка
                 // TCombobox

procedure TForm1.Button1Click(Sender: TObject);
begin
 {загружаем раскладку, соответствующую числу в MyLangID}
 if LoadKeyboardLayout(LangID, KLF_ACTIVATE) = 0
 then
    ShowMessage('Ошибка загрузки раскладки');
 GetLayoutList;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  {активация выделенной раскладки}
  if
ActivateKeyboardLayout(StrToInt64(ListBox1.Items[Listbox1.Itemindex]),
      KLF_REORDER) = 0) then
   ShowMessage('Ошибка активации выделенной раскладки');
  {Очистка и обновление списка раскладок}
  GetLayoutList;
end;

procedure TForm1.GetLayoutList;
begin
  ListBox1.Clear;
  MyListIndex := 0;
  While (List[MyListIndex] <> 0) do
  begin
    List[MyListIndex]:=0;
    Inc(MyListIndex);
  end;
  {получение списка раскладок клавиатуры}
  GetKeyboardLayoutList(MAX_HKL, List);
  {создание буфера для текущего имени раскладки}
  GetMem(pwszKLID, KL_NAMELENGTH);
  {получение имени активной раскладки}
  GetKeyboardLayoutName(pwszKLID);
  StatusBar1.SimpleText:= 'Имя активной раскладки: ' + pwszKLID;
  {получение идентификатора кодовой страницы}
  StaticText1.Caption:=IntTostr(GetACP);
```

```
  {очистка буфера для текущего имени раскладки}
  FreeMem(pwszKLID);
  {отображение всех текущих раскладок}
  MyListIndex := 0;
  while (List[MyListIndex] <> 0) do
  begin
     ListBox1.Items.Add(IntToStr(List[MyListIndex]));
     Inc(MyListIndex);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
{выгрузить раскладку}
if not
UnloadKeyboardLayout(StrToInt64(ListBox1.Items[Listbox1.ItemIndex])) then
   ShowMessage('Ошибка выгрузки раскладки');
 GetLayoutList;
end;
```

Пример, сокращенный код которого представлен в листинге 6.8, демонстрирует использование функции ввода `DragDetect` Windows API по перехвату событий мыши и слежению за ее перемещением.

Полный исходный код данного примера находится на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex11.

Листинг 6.8. Использование функции ввода `DragDetect`

```
procedure TForm1.Panel1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  XY_coordinates,
  UV_coordinates: TPoint; // Старые координаты
begin
  {координаты прямоугольника}
  XY_coordinates.X:= Panel1.Width;
  XY_coordinates.Y:= Panel1.Height;
  {детектирование перемещений мыши над панелью при нажатии кнопки мыши}
  while(DragDetect(Panel1.Handle,XY_coordinates)) do
  begin
       {позиция над панелью}
        GetCursorPos(UV_coordinates);
        Windows.ScreenToClient(Panel1.Handle, UV_coordinates);
```

```
        {отобразить координаты}
        Edit1.Text:= IntToStr(UV_coordinates.X);
        Edit2.Text:= IntToStr(UV_coordinates.Y);
        {нарисовать}
        SetPixel(GetDc(Panel1.Handle), UV_coordinates.x,
                 UV_coordinates.Y,
                 ColorGrid1.ForegroundColor);
  end
end;
```

И последний пример данного раздела, полный исходный код которого находится на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex12, демонстрирует использование функции ввода `SetCaretBlinkTime`, управляющей интервалом мерцания каретки.

Заканчивая рассказ о функциях ввода, можно сказать, что если вашему приложению будут необходимы какие-либо специализированные способы работы с клавиатурой и мышью, то вы на практике сможете оценить возможности приведенных функций.

Строковые функции и функции атомов

Несмотря на то, что язык Object Pascal обеспечивает разработчика богатым набором функций для работы со строками, функции Windows позволяют помимо этого получить доступ к внутренним строковым таблицам, используемым системой для сообщений об ошибках.

Приложения Windows могут хранить строки длиной до 255 символов в так называемой таблице атомов. Каждый процесс имеет доступ к глобальной таблице атомов и использует свою собственную, локальную таблицу.

Глобальная таблица атомов имеет максимум 37 элементов и доступна другим процессам. Локальная таблица атомов доступна только своему процессу, она также имеет 37 элементов, но может быть расширена, если она была создана функцией `InitAtomTable`. Строка добавляется к таблице с помощью функций `AddAtom` или `GlobalAddAtom` и удаляется функциями `DeleteAtom` или `GlobalDeleteAtom`. Каждая строка (или атом) идентифицируется своим уникальным номером в таблице.

Таблицы атомов обычно используются для хранения строк, но могут также содержать 16-битные целые числа, которые создаются функцией `MakeIntAtom`, создающей указатель на строку с конечным нулем, совместимую с функциями `AddAtom` and `GlobalAddAtom`. Эти целые числа интерпретируются как строки. Количество атомов для целых чисел лежит в диапазоне от 1 до 49151 (в ше-

стнадцатеричном представлении от $0001 до $BFFF), тогда как для строковых значений — от 49 152 до 65 535 (от $C000 до $FFFF соответственно). Атом с номером 0 используется как флаг ошибки.

Наиболее широко глобальные таблицы атомов используются для обмена данными в приложениях DDE. Прежде чем передать строку данных в вызывающее приложение, DDE сохраняет строку в глобальной таблице атомов и после этого передает 16-битный номер атома. DDE-сервер по этому номеру находит строку в глобальной таблице. Локальная таблица атомов существует только в течение определенного процесса. Глобальная таблица существует на протяжении всего сеанса Windows до его завершения или перезагрузки. Атом в глобальной таблице может сохраняться и после того, как завершится создавший ее процесс. Однако в силу того, что пространство для строк в локальных и глобальной таблицах ограничено, обычно созданные атомы удаляются после завершения работы с таблицей. Это особенно важно для глобальной таблицы, поскольку ее могут использовать и другие приложения.

Каждый атом в таблице атомов имеет свой порядковый номер. Если AddAtom или GlobalAddAtom делают попытку добавить строку, которая уже существует в таблице, тоновый элемент не создается. Порядковый номер существующего элемента будет увеличен и порядковый номер элемента, содержащий строку, будет возвращен в виде результата. Если добавляется атом, который ранее не существовал, то создается новый элемент, и его номер устанавливается равным 1. Функции DeleteAtom и GlobalDeleteAtom уменьшают порядковый номер на единицу и проверяют его; если он равен 0, то атом удаляется из таблицы. Исключением из этого правила являются целочисленные атомы. Целые числа, хранящиеся в таблице атомов, не имеют порядкового номера и удаляются из таблицы немедленно. К сожалению, нельзя напрямую определить порядковый номер атома. Чтобы убедиться, что атом удален из таблицы, приложение должно продолжить процесс удаления до тех пор, пока функция DeleteAtom или GlobalDeleteAtom не вернет ошибку.

В табл. 6.17 представлены соответствующие функции API по работе со строками и атомами.

Таблица 6.17. *Строковые и атомарные функции*

Имя	Описание
AddAtom	Добавляет строку в локальную таблицу атомов
CharLower	Преобразует строку символов в строчные буквы
CharLowerBuff	Преобразует символы из буфера в строчные буквы
CharNext	Возвращает указатель на следующий символ строки

Таблица 6.17 *(продолжение)*

Имя	Описание
CharPrev	Возвращает указатель на предыдущий символ строки
CharToOem	Преобразует строку из набора символов текущей региональной установки в набор символов OEM
CharToOemBuff	Преобразует заданное количество символов в строке из набора символов текущей региональной установки в набор символов OEM
CharUpper	Преобразует строку символов в прописные буквы
CharUpperBuff	Преобразует символы из буфера в прописные буквы, используя идентификатор текущей региональной установки
CompareString	Сравнивает две символьные строки, используя указанную региональную установку
DeleteAtom	Удаляет атом из локальной таблицы атомов
EnumSystemCodePages	Перечисляет кодовые страницы, которые установлены или поддерживаются в системе
EnumSystemLocales	Перечисляет региональные установки, которые поддерживаются в системе
FindAtom	Выполняет поиск имени атома, которое соответствует указанной строке, в локальной таблице атомов без учета регистра
FormatMessage	Форматирует строку сообщения, используя определение сообщения
GetACP	Возвращает идентификатор кодовой страницы ANSI
GetAtomName	Осуществляет выборку имени атома, которое хранится в локальной таблице атомов. Имя копируется в соответствующий буфер
GetCPInfo	Находит информацию о любой установленной или доступной действительной кодовой странице
GetCPInfoEx	Находит расширенную информацию о любой установленной или доступной действительной кодовой странице
GetDateFormat	Форматирует дату в виде строки с датой, характерной для заданной региональной установки
GetTimeFormat	Форматирует время, используя формат, характерный для региональной установки либо предоставляемый в виде шаблона формата
GlobalAddAtom	Сохраняет строку в глобальной таблице атомов приложения либо увеличивает счетчик ссылок на существующий атом

Таблица 6.17 *(окончание)*

Имя	Описание
GlobalDeleteAtom	Удаляет атом из глобальной таблицы атомов
GlobalFindAtom	Ищет указанную строку в глобальной таблице атомов без учета регистра
GlobalGetAtomName	Осуществляет выборку имени атома, которое связано с данным глобальным атомом, из глобальной таблицы атомов
InitAtomTable	Задает определенное количество элементов верхнего уровня в локальной таблице атомов
IsCharAlpha	Проверяет, является ли указанный символ буквенным
IsCharAlphaNumeric	Проверяет, является ли указанный символ буквенно-цифровым
IsCharLower	Проверяет, является ли указанный символ строчной буквой
IsCharUpper	Проверяет, является ли указанный символ прописной буквой
lstrcat	Сцепляет две нуль-терминальные строки
lstrcmp	Сравнивает две нуль-терминальные строки с учетом регистра, используя текущий язык
lstrcmpi	Сравнивает две нуль-терминальные строки без учета регистра, используя текущий язык
lstrcpy	Копирует одну нуль-терминальную строку в другую (по другому адресу)
lstrlen	Возвращает длину нуль-терминальной строки
MakeIntAtom	Создает целочисленный атом
OemToChar	Преобразует строку из набора OEM в набор символов, характерный для текущей региональной установки
OemToCharBuff	Преобразует данные в буфере из набора OEM в набор символов, характерный для текущей региональной установки
ToAscii	Преобразует код виртуальной клавиши и состояние клавиатуры в соответствующие символы Windows
wvsprintf	Форматирует данные с использованием указанной строки формата и размещает результат в символьном буфере

Для понимания работы этих функций рассмотрим некоторые примеры.

Пример, сокращенно представленный в листинге 6.9, иллюстрирует работу с локальной таблицей атомов. Исходный код примера можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex13.

Листинг 6.9. Работа с локальной таблицей атомов

```
procedure TForm1.Button1Click(Sender: TObject);
var
 MyAtom: Atom;    // Возвращаемый номер атома
 TextTest: PChar; // Строка для поиска
 AtomTest: Atom;  // Найденный атом
begin
 {добавить атом в локальную таблицу атомов}
 MyAtom := AddAtom(PChar(Edit1.Text));
 {поискать в локальной таблице атомов}
 AtomTest := FindAtom(PChar(Edit1.Text));
 Label1.Caption := 'Номер атома = ' + IntToStr(Atomtest);
 {найденный текст}
 TextTest := StrAlloc(256);
 GetAtomName(MyAtom, TextTest, 256);
 Label2.Caption := 'Найденный текст: ' + string(TextTest);
 DeleteAtom(MyAtom);
end;
```

В примере, представленном в листинге 6.10, мы воспользуемся функцией `EnumCodePageProc` для получения номеров установленных или поддерживаемых кодовых страниц. Исходный код примера можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex14.

Листинг 6.10. Получение номеров кодовых страниц

```
Form1: TForm1;
{объявляем функцию обратного вызова}
function EnumCodePageProc(AnotherCodePage: PChar): Integer; stdcall;

implementation
{$R *.dfm}

{функция обратного вызова будет вызвана столько раз, сколько кодовых
страниц будет найдено}
function EnumCodePageProc(AnotherCodePage: PChar):integer;
begin
```

```
  {добавляем номер кодовой страницы в окно вывода}
  Form1.Memo1.Lines.Add(string(AnotherCodePage));
  Result := 1;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  MyFlag: Integer; {флаг, выбираемый с помощью переключателей на форме}
begin
  {начальные установки}
  Memo1.Clear;
  MyFlag := CP_SUPPORTED; {устанавливаем начальное значение флага}
  if RadioGroup1.ItemIndex = 0 then
     MyFlag := CP_INSTALLED;
  {перебор кодовых страниц в системе, соответствующих
   установке флага Myflag}
  if not EnumSystemCodePages(@EnumCodePageProc, MyFlag) then
     ShowMessage('Ошибка получения кодовых страниц');

  StatusBar1.SimpleText := 'Кодовых страниц найдено: ' +
                           IntToStr(Memo1.Lines.Count);
end;
```

Следующий пример иллюстрирует работу функций `GetCPInfo` и `GetCPInfoEx` для получения информации о кодовой странице. Исходный код примера можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex15.

Последний пример данного раздела, представленный в листинге 6.11, демонстрирует использование функции `IsCharAlphaNumeric` для отсеивания при вводе строки символов, не являющихся буквенно-цифровыми. Исходный код данного примера можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex16.

Листинг 6.11. Использование функции `IsCharAlphaNumeric`

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
var
  Ch: char;
begin
 // Проверка, является ли указанный символ буквенно-цифровым
 if IsCharAlphaNumeric(Key)  then
 begin
```

```
    Edit1.ReadOnly := False;
    Edit1.Text := Edit1.Text + Key;
    Edit1.ReadOnly := True;
    Edit1.SelStart := Length(Edit1.Text);
 end
 else
    beep;
end;
```

Функции работы с буфером обмена

Трудно представить эффективную работу в многозадачной среде без использования буфера обмена. Windows API имеет собственный инструментарий для управления буфером обмена.

Буфер обмена — это непросто выделяемая операционной системой память для обмена данными, а совокупность специальных "корзин", имеющих указатели (дескрипторы) на информацию, хранящуюся в специальном формате.

Буфер обмена поддерживает некоторое количество предопределенных форматов для хранения информации различных типов, таких как графика и текст, хотя любое приложение может определять свой собственный формат. Предопределенные форматы распознаются функциями `GetClipboardData` и `SetClipboardData`. Для помещения информации в буфер приложение сначала должно открыть его функцией `OpenClipboard`, затем предварительно очистить буфер функцией `EmptyClipboard`, которая удаляет всю информацию из буфера в любом формате и назначает буфер конкретному окну, передавая параметр `hWndNewOwner`, установленный в функции `OpenClipboard`. Затем приложение вызывает функцию `SetClipboardData`, передавая ей флаг, указывающий формат данных, и указатель на сами данные. Функция `CloseClipboard` закрывает буфер и завершает работу с ним. Приложение может получить данные из буфера с помощью функции `GetClipboardData`, передавая ей флаг формата запрашиваемых данных. Если данные в буфере существуют в запрашиваемом формате, то будет возвращен указатель на них. При помещении приложением данных в буфер желательно, чтобы они находились там во всех возможных форматах. Это нужно, чтобы наибольшее количество приложений могло ими воспользоваться. Например, если текстовый процессор поместит данные в буфер только в собственном формате, то они будут доступны только запущенным экземплярам этого процессора. Если этот процессор поместит данные в буфер дополнительно к собственному формату еще и в форматах `CF_TEXT` и `CF_UNICODETEXT`, то эти данные будут доступны другим Windows-приложениям, таким как Блокнот или MS Word.

Windows обеспечивает преобразование различных предопределенных форматов один в другой. Это преобразование происходит "на лету" и прозрачно для

приложений. В табл. 6.18 показаны наименования форматов, их возможные варианты конвертирования и поддержка операционными системами Windows.

Таблица 6.18. *Форматы данных буфера обмена*

Формат	Конвертируется в	Поддержка Windows
CF_BITMAP	CF_DIB	Windows 95/98/ME, NT/2000/XP
CF_DIB	CD_BITMAP	Windows 95/98/ME, NT/2000/XP
CF_DIB	CF_ PALETTE	Windows 95/98/ME, NT/2000/XP
CF_METAFILEPICT	CF_ENHMETAFILE	Windows 95/98/ME, NT/2000/XP
CF_ENHMETAFILE	CF_METAFILEPICT	Windows 95/98/ME, NT/2000/XP
CF_OEMTEXT	CF_UNICODETEXT	Windows NT/2000/XP
CF_OEMTEXT	CF_TEXT	Windows 95/98/ME, NT/2000/XP
CF_TEXT	CF_OEMTEXT	Windows 95/98/ME, NT/2000/XP
CF_TEXT	CF_UNICODETEXT	Windows NT/2000/XP
CF_UNICODETEXT	CF_OEMTEXT	Windows NT/2000/XP
CF_UNICODETEXT	CF_TEXT	Windows NT/2000/XP

В табл. 6.19 приведены функции работы с буфером обмена.

Таблица 6.19. *Функции работы с буфером обмена*

Имя	Описание
ChangeClipboardChain	Удаляет окно просмотра буфера обмена из цепочки
CloseClipboard	Закрывает буфер обмена для чтения или записи его содержимого
CountClipboardFormats	Возвращает количество форматов, воспроизводимых текущим владельцем буфера обмена
EmptyClipboard	Удаляет содержимое буфера обмена и сопутствующие данные
EnumClipboardFormats	Перечисляет все форматы буфера обмена, воспроизводимые его текущим владельцем
GetClipboardData	Извлекает дескриптор содержимого буфера обмена

Таблица 6.19 *(окончание)*

Имя	Описание
GetClipboardFormatName	Указывает имя зарегистрированного формата буфера обмена
GetClipboardOwner	Возвращает дескриптор окна владельца буфера обмена
GetClipboardViewer	Возвращает дескриптор окна первой программы просмотра буфера обмена в цепочке
GetOpenClipboardWindow	Определяет дескриптор окна, для которого в данный момент открыт буфер обмена
GetPriorityClipboardFormat	Проверяет доступность в буфере обмена требуемых форматов данных
IsClipboardFormatAvailable	Указывает, содержит ли буфер обмена данные в определенном формате
OpenClipboard	Открывает буфер обмена для чтения и записи его содержимого
RegisterClipboardFormat	Регистрирует в Windows имя нового формата буфера обмена
SetClipboardData	Помещает данные в буфер обмена для данного формата
SetClipboardViewer	Помещает конкретное окно в начало цепочки окон просмотра буфера обмена

Продемонстрируем работу функций работы с буфером обмена на примерах.

Используя функции работы с буфером обмена, можно, например, определить поддерживаемые форматы буфера обмена его текущего владельца. Исходный код этого примера, приведенного в листинге 6.12, можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex17. Результат выполнения программы представлен на рис. 6.5.

Листинг 6.12. Определение текущего формата буфера обмена

```
procedure TForm1.Timer1Timer(Sender: TObject);
const
{массив, содержащий все предопределенные имена форматов буфера обмена}
PredefinedClipboardNames: array[1..17] of string =
            ('CF_TEXT', 'CF_BITMAP',
```

```
            'CF_METAFILEPICT', 'CF_SYLK', 'CF_DIF',
            'CF_TIFF', 'CF_OEMTEXT', 'CF_DIB',
            'CF_PALETTE', 'CF_PENDATA', 'CF_RIFF',
            'CF_WAVE', 'CF_UNICODETEXT',
            'CF_ENHMETAFILE', 'CF_HDROP', 'CF_LOCALE',
            'CF_MAX');
var
FormatID: UINT; {для ID-формата буфера обмена}
FormatName: array[0..255] of char; {для имени формата буфера обмена}
Len: Integer; {для длины формата буфера обмена}
begin
{очищаем окно вывода}
ListBox1.Items.Clear;
{подсчитываем и выводим количество форматов, поддерживаемых текущим
 владельцем буфера обмена}
Label1.Caption :=
  'Всего форматов доступно: '+IntToStr(CountClipboardFormats);
{открываем буфер обмена}
OpenClipboard(0);
{пытаемся считать первый из доступных форматов}
FormatID := EnumClipboardFormats(0);
{пытаемся считать остальные форматы}
while (FormatID<>0 ) do
begin
 {получаем имя формата буфера обмена, заметим, что будет возвращено
 имя зарегистрированного, а не предопределенного формата}
 Len := GetClipboardFormatName(FormatID,FormatName,255);

 {если длина равна нулю, то это предопределенный формат}
 if Len = 0 then
  ListBox1.Items.Add(PredefinedClipboardNames[FormatID]+
   ' (Предопределенный)'+
   ' [' + IntToStr(FormatID)+ ']')
 else
  {иначе это зарегистрированное имя формата}
  ListBox1.Items.Add(FormatName+' [' + IntToStr(FormatID)+ ']');
 {получаем ID следующего формата буфера обмена}
 FormatID:=EnumClipboardFormats(FormatID);
end;
{обязательно закрываем буфер}
CloseClipboard;
end;
```

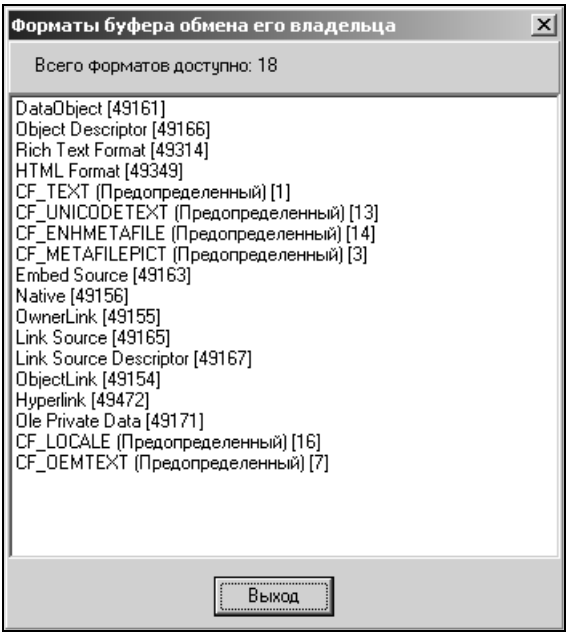


Рис. 6.5. Определение текущего формата буфера обмена

Функции системной информации

Некоторые приложения могут запрашивать различные параметры о самой работающей операционной системе и ее устройствах. Далее рассмотрены функции, возвращающие такие характеристики, как имя компьютера, системное время, версия Windows, переменные окружения и др. Стоит отметить, что некоторые функции помимо получения различной информации могут также ее изменять.

В табл. 6.20 приведены функции системной информации, с помощью которых ваши приложения смогут получить доступ ко многим системным свойствам.

Таблица 6.20. *Функции системной информации*

Имя	Описание
`DnsHostnameToComputerName`	Преобразует имя компьютера из формата DNS в формат NetBIOS
`ExpandEnvironmentStrings`	Расширяет строки переменных окружения и заменяет их определенными для них значениями

Таблица 6.20 *(продолжение)*

Имя	Описание
FreeEnvironmentStrings	Очищает блок переменных окружения, указатель на который возвращает функция GetEnvironment-Strings
GetCommandLine	Возвращает командную строку процесса
GetComputerName	Возвращает NetBIOS-имя локального компьютера, которое существовало в момент запуска системы
GetComputerNameEx	Возвращает NetBIOS- или DNS-имя локального компьютера, которое существовало в момент запуска системы
GetDiskFreeSpaceEx	Позволяет получить информацию о размере доступного дискового пространства тома
GetDriveType	Возвращает тип накопителя по заданному имени корневого пути
GetEnvironmentStrings	Возвращает указатель на блок переменных окружения процесса
GetEnvironmentVariable	Возвращает значение, связанное с переменной окружения процесса
GetLocaleInfo	Возвращает информацию о региональной установке
GetLocalTime	Возвращает местное время из структуры типа SYSTEMTIME. Это время настраивается на летнее для конкретного временного пояса
GetLogicalDrives	Возвращает битовую маску логических дисков, которые доступны в настоящий момент
GetLogicalDriveStrings	Заполняет буфер строками, в которых перечисляются действительные логические диски в системе
GetStartupInfo	Возвращает информацию, имеющую отношение к тому, каким образом вызывающий данную функцию процесс был установлен его родительским процессом
GetSystemDefaultLangID	Возвращает идентификатор языка, используемого в системе по умолчанию
GetSystemDefaultLCID	Возвращает идентификатор региональной установки, используемой в системе по умолчанию
GetSystemDirectory	Получает имя системного каталога Windows
GetSystemInfo	Выбирает информацию об аппаратных средствах, установленных в системе

Таблица 6.20 *(продолжение)*

Имя	Описание
GetSystemTime	Осуществляет выборку данных из структуры типа SYSTEMTIME, в которой время сообщается в формате UTC (Universal Time Coordinated — универсальное скоординированное время)
GetSystemTimeAsFileTime	Возвращает системное время в формате, понятном текущей файловой системе
GetTimeZoneInformation	Осуществляет выборку данных о временном поясе
GetUserDefaultLangID	Возвращает идентификатор языка для используемой по умолчанию региональной установки текущего пользователя
GetUserDefaultLCID	Возвращает идентификатор региональной установки для используемой по умолчанию региональной установки текущего пользователя
GetUserName	Получает имя пользователя текущей нити
GetUserNameEx	Получает имя пользователя текущей нити в указанном формате
GetVersionEx	Осуществляет выборку информации, полностью определяющую версию данной системы, в том числе основной и дополнительный номера версии, а также номер создания системы
GetVolumeInformation	Получает метку тома и данные о файловой системе этого тома на указанном накопителе
GetWindowsDirectory	Осуществляет выборку каталога, в котором установлена операционная система Windows
IsProcessorFeaturePresent	Определяет, доступны ли свойства центрального процессора
SetComputerName	Изменяет NetBIOS-имя компьютера. Изменение вступит в силу только после перезапуска системы
SetComputerName	Изменяет NetBIOS- и DNS-имя компьютера. Изменение вступит в силу только после перезапуска системы
SetEnvironmentVariable	Изменяет значение существующей переменной окружения либо создает новую с указанным значением
SetLocaleInfo	Изменяет данные, имеющие отношение к региональной установке
SetLocalTime	Задает текущие местное время и дату

Таблица 6.20 *(окончание)*

Имя	Описание
SetSystemTime	Изменяет системное время в соответствии со значениями, указанными в структуре SYSTEMTIME
SetSystemPowerState	Временно завершает работу системы
SetTimeZoneInformation	Задает параметры текущего временного пояса
SetVolumeLabel	Задает метку тома на указанном накопителе
SystemParametersInfo	Запрашивает и задает параметры системы
VerLanguageName	Осуществляет выборку хранимого названия языка из идентификатора языка

Функции системной информации весьма полезны при разработке приложений, активно использующих информацию о системной конфигурации.

Пример, представленный в листинге 6.13, демонстрирует работу функций ExpandEnvironmentStrings и GetCommandLine, с помощью которых находится текущий системный каталог, считывается значение переменной окружения и выдается командная строка текущего процесса. Исходный код примера находится на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex18. Результат выполнения программы представлен на рис. 6.6.

Листинг 6.13. Получение системной информации

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ExpandedStr: array[0..255] of char; // holds the expanded environment
string
begin
  {считываем значение переменной окружения}
  ExpandEnvironmentStrings('Каталог для временных файлов: %TEMP%',
                           ExpandedStr, SizeOf(ExpandedStr));
  {выводим полученное значение переменной}
  Label1.Caption := StrPas(ExpandedStr);
  {повторяем те же действия для другой переменной}
   ExpandEnvironmentStrings('Системный каталог: %windir%',
                            ExpandedStr, SizeOf(ExpandedStr));
                            Label2.Caption := StrPas(ExpandedStr);
   {получаем командную строку процесса}
   Label3.Caption := 'Командная строка текущего процесса:' +
                     StrPas(GetCommandLine);
end;
```

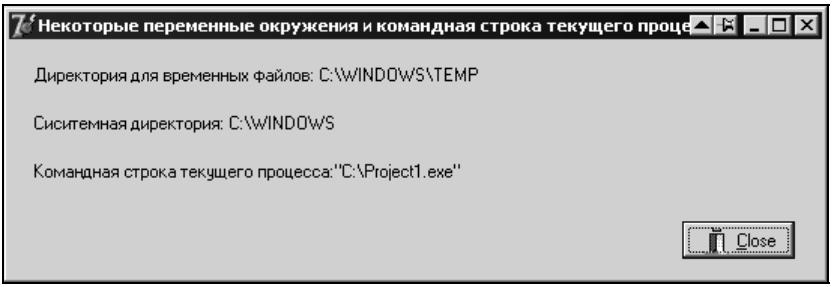
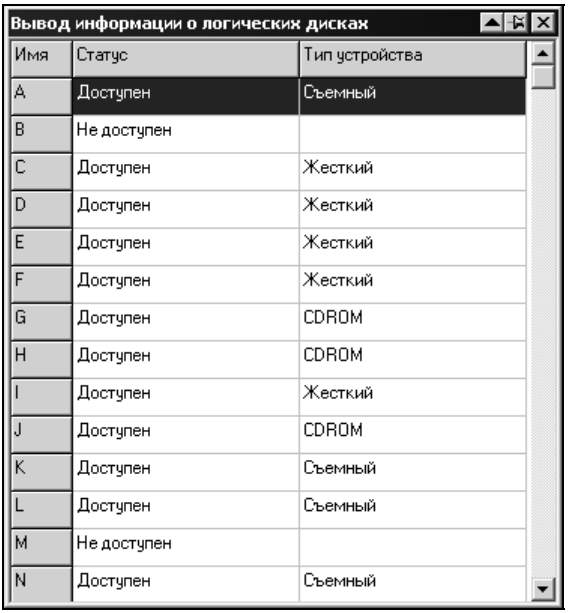


Рис. 6.6. Получение системной информации

Пример, сокращенно представленный в листинге 6.14, позволяет получить информацию о логических дисках, используемых в системе. Исходный код примера можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex19. Результат выполнения программы представлен на рис. 6.7.



Имя	Статус	Тип устройства
A	Доступен	Съемный
B	Не доступен	
C	Доступен	Жесткий
D	Доступен	Жесткий
E	Доступен	Жесткий
F	Доступен	Жесткий
G	Доступен	CDROM
H	Доступен	CDROM
I	Доступен	Жесткий
J	Доступен	CDROM
K	Доступен	Съемный
L	Доступен	Съемный
M	Не доступен	
N	Доступен	Съемный

Рис. 6.7. Получение информации о логических дисках

Листинг 6.14. Получение информации о логических дисках в системе

```
procedure TForm1.FormCreate(Sender: TObject);
var
AvailableDrives: DWord; {для хранения битовой маски доступного устройства}
Counter: Integer; {счетчик цикла}
```

```
DrivePath: array[0..3] of Char; {для хранения имени устройства}
begin
{задаем ширину колонок}
StringGrid1.ColWidths[0] := 40;
StringGrid1.ColWidths[1] := 150;
StringGrid1.ColWidths[2] := 150;
{задаем названия столбцов таблицы}
StringGrid1.Cells[0, 0] := 'Имя';
StringGrid1.Cells[1, 0] := 'Статус';
StringGrid1.Cells[2, 0] := 'Тип устройства';

{получим битовую маску логических дисков,
которые доступны в настоящий момент}
AvailableDrives := GetLogicalDrives;
{цикл по всем 26 доступным дисковым устройствам}
for Counter := 0 to 25 do
begin
 {отображаем имя устройства}
 StringGrid1.Cells[0, Counter + 1] := Char(Ord('A') + Counter);
 {если устройство доступно, то (используем битовую маску) ...}
 if LongBool(AvailableDrives and ($0001 shl Counter)) = True then
 begin
  if StringGrid1.RowCount>=2 then
    StringGrid1.RowCount := StringGrid1.RowCount +1;
  {выводим текст, если устройство доступно}
  StringGrid1.Cells[1, Counter + 1] := 'Доступен';
  {присвоим значение параметру DrivePath для функции GetDrivePath}
  StrpCopy(DrivePath,Char(Ord('A') + Counter));
  StrCat(DrivePath, ':\');
  {получим и отобразим тип устройства}
  case GetDriveType(DrivePath) of
   DRIVE_UNKNOWN:
     StringGrid1.Cells[2, Counter + 1] := 'Нет данных';
   DRIVE_NO_ROOT_DIR:
     StringGrid1.Cells[2, Counter + 1] := 'Root does not exist';
   DRIVE_REMOVABLE: StringGrid1.Cells[2, Counter + 1] := 'Съемный';
   DRIVE_FIXED: StringGrid1.Cells[2, Counter + 1] := 'Жесткий';
   DRIVE_REMOTE: StringGrid1.Cells[2, Counter + 1] := 'Сетевой';
   DRIVE_CDROM: StringGrid1.Cells[2, Counter + 1] := 'CDROM';
   DRIVE_RAMDISK:
     StringGrid1.Cells[2, Counter + 1] := 'Виртуальный';
  end;
 end
```

```
 else
  {если устройство недоступно, то выводим}
  StringGrid1.Cells[1, Counter + 1] := 'Недоступен';
end;
end;
```

Пример, сокращенно представленный в листинге 6.15, демонстрирует возможность функций системной информации не только получать различные системные свойства, но и изменять их. Так в данном примере программа с помощью функции `SystemParametersInfo` получает размеры кнопок заголовка окна и увеличивает или уменьшает их в два раза.

Стоит обратить внимание на эту функцию, т. к. она имеет очень большое количество параметров, отвечающих за различные системные свойства. Используя данные параметры, программист получает значительные возможности для манипулирования системой. В SDK можно найти подробное описание всех параметров.

Исходный код примера можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex20. Результат выполнения программы до и после изменения высоты кнопок представлен на рис. 6.8.

Листинг 6.15. Изменение некоторых системных свойств с помощью функции `SystemParametersInfo`

```
var
 MyNCM: TNonClientMetrics; // Старая информация о метриках
 OriginalWidth,
 OriginalHeight: Integer;  // Старая информация о размере кнопок

procedure TForm1.Button1Click(Sender: TObject);
begin
 {инициализация}
 MyNCM.cbSize := SizeOf(TNonClientMetrics);
 {Получение системной информации}
 SystemParametersInfo(SPI_GetNonClientMetrics,
                      SizeOf(TNonClientMetrics),@MyNCM, 0);
 {увеличиваем кнопки в 2 раза}
 MyNCM.iCaptionWidth := MyNCM.iCaptionWidth * 2;
 MyNCM.iCaptionHeight := MyNCM.iCaptionHeight * 2;
 {применяем новый размер}
 SystemParametersInfo(SPI_SetNonClientMetrics, SizeOf(TNonClientMetrics),
                      @MyNCM, SPIF_SENDWININICHANGE);
end;
```

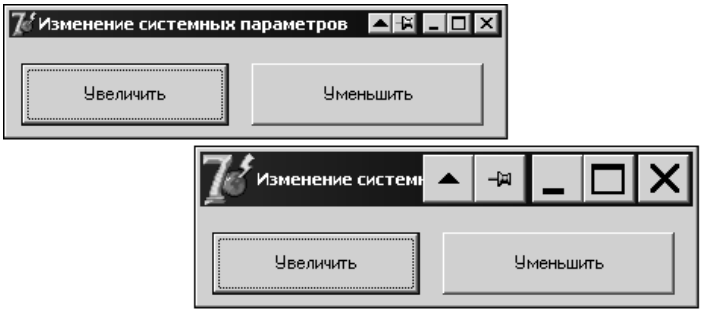


Рис. 6.8. Увеличение/уменьшение размеров кнопок

Рассмотрим пример, демонстрирующий возможности функции `SystemParametersInfo`. В примере, представленном в листинге 6.16, мы покажем возможность изменять обои рабочего стола из приложения.

Исходный код примера можно найти на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex21. Результат выполнения программы представлен на рис. 6.9.

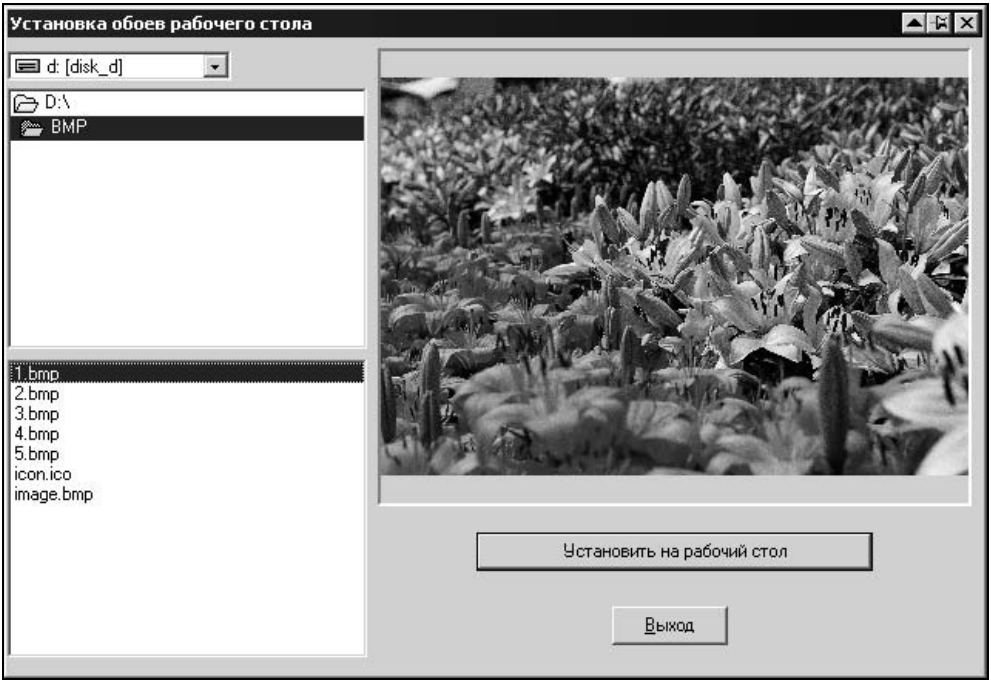


Рис. 6.9. Установка обоев рабочего стола

Листинг 6.16. Установка обоев рабочего стола

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 {пытаемся установить выбранный рисунок на рабочий стол}
 if FileListBox1.ItemIndex>-1 then
    SystemParametersInfo(SPI_SETDESKWALLPAPER, 0,
                         PChar(FileListBox1.FileName),
                         SPIF_SENDCHANGE);
end;
```

Функции каретки, курсора и иконок

Каретка, курсор мыши и иконки (пиктограммы) представляют собой небольшие графические объекты, управляемые операционной системой. Вряд ли без них работа с оконным интерфейсом будет удобна. Windows располагает группой функций, позволяющих напрямую управлять этими объектами.

В табл. 6.21 приведены функции каретки, курсора и иконок.

Таблица 6.21. *Функции каретки, курсора и иконок*

Имя	Описание
CopyIcon	Копирует иконку из другого модуля в текущий модуль
CopyImage	Создает новую иконку, курсор или растровое изображение с атрибутами другого изображения
CreateBitmap	Позволяет создать на основе массива двоичных значений цвета растровое изображение с заданной шириной, высотой и форматом цвета (с заданным числом цветовых плоскостей и битов на пиксел)
CreateCaret	См. табл. 6.16
CreateCursor	См. табл. 6.16
CreateIcon	Создает иконку с указанным размером, цветами и двоичными шаблонами
CreateIconFromResource	Создает иконку или курсор из двоичных данных ресурса с описанием иконки
CreateIconFromResourceEx	Аналогична функции CreateIconFromResource за исключением того, что в ней можно указывать размеры изображения

Таблица 6.21 *(окончание)*

Имя	Описание
CreateIconIndirect	Создает иконку или курсор по данным структуры ICONINFO
DestroyCaret	Уничтожает объект курсора каретки
DestroyCursor	Уничтожает объект курсора мыши
DestroyIcon	Уничтожает иконку, созданную с помощью CreateIcon, и освобождает память, занимаемую иконкой
DrawIcon	Выводит иконку в клиентской области окна, принадлежащего указанному контексту устройства
DrawIconEx	Аналогична функции DrawIcon, кроме того, позволяет выполнить определенные операции над изображением курсора
ExtractIcon	Возвращает дескриптор иконки из указанного файла
ExtractIconEx	Возвращает дескриптор большой или маленькой иконки из указанного файла
GetCursor	См. табл. 6.16
GetIconInfo	Позволяет получить информацию о иконке или курсоре
HideCaret	См. табл. 6.16
LoadCursor	См. табл. 6.16
LoadCursorFromFile	См. табл. 6.16
LoadIcon	Загружает иконку из файла описания ресурса
LookupIconIdFromDirectory	Выполняет поиск в данных иконки или курсора такого экземпляра иконки или курсора, который лучше всего подходит для текущего устройства отображения
LookupIconIdFromDirectoryEx	Аналогична LookupIconIdFromDirectory, кроме того, позволяет указать желаемые размеры иконки или курсора
SetCursor	См. табл. 6.16
SetSystemCursor	Устанавливает системный курсор
ShowCaret	См. табл. 6.16
ShowCursor	См. табл. 6.16

После того как мы кратко ознакомились с этими функциями, покажем работу некоторых из них на конкретных примерах.

В листинге 6.17 приведен пример, выводящий иконку текущего приложения. Пример демонстрирует использование функций `CopyIcon` и `DrawIcon`. Исходный код примера находится на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex22.

Листинг 6.17. Получение иконки приложения

```
procedure TForm1.Button1Click(Sender: TObject);
var
 IconCopy: HICON; {для хранения дескриптора копируемой иконки}
begin
 {создаем копию иконки приложения...}
 IconCopy := CopyIcon(Application.Icon.Handle);
 {...и показываем ее}
 DrawIcon(PaintBox1.Canvas.Handle, (PaintBox1.Width div 2)-16,
         (PaintBox1.Height div 2)-16, IconCopy);
end;
```

Следующий пример, сокращенный код которого приведен в листинге 6.18, демонстрирует работу функции `GetIconInfo`, получающей информацию о системных курсорах и иконках. Полный исходный код примера находится на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex23. Результат выполнения программы представлен на рис. 6.10.

Листинг 6.18. Получение информации о системных курсорах и иконках

```
type
  TIconTypes = array[0..8] of PAnsiChar;
  TCursorTypes = array[0..12] of PAnsiChar;

const
  IconTypes: TIconTypes = (IDI_APPLICATION,IDI_ASTERISK,IDI_ERROR,
                           IDI_EXCLAMATION,IDI_HAND,IDI_INFORMATION,
                           IDI_QUESTION,IDI_WARNING,IDI_WINLOGO);

  CursorTypes: TCursorTypes = (IDC_ARROW,IDC_IBEAM,IDC_WAIT,IDC_CROSS,
                               IDC_UPARROW,IDC_SIZENWSE,IDC_SIZENESW,
                               IDC_SIZEWE,IDC_SIZENS,IDC_SIZEALL,
                               IDC_NO,IDC_APPSTARTING,IDC_HELP);
```

```
procedure TForm1.ComboBox1Change(Sender: TObject);
var
  TheIcon: HICON; {для хранения дескриптора иконки}
  TheCursor: HCURSOR; {для хранения дескриптора курсора}
  TheIconInfo: TIconInfo; {для хранения информации иконки или курсора}
begin
  {стираем предыдущие изображения, если есть}
  Image1.Canvas.Brush.Color:=clBtnFace;
  Image1.Canvas.Fillrect(Image1.Canvas.Cliprect);
  if TheIconInfo.hbmMask<>0 then DeleteObject(TheIconInfo.hbmMask);
  if TheIconInfo.hbmColor<>0 then DeleteObject(TheIconInfo.hbmColor);
  {если переключателем RadioButton выбраны иконки, то
   выдаем информацию об иконке, выбранной в ComboBox...}
  if RadioButton1.Checked then
  begin
    {загружаем выбранную системную иконку}
    TheIcon:=LoadIcon(0, IconTypes[ComboBox1.ItemIndex]);
    {заполняем информационную структуру об этой иконке}
    GetIconInfo(TheIcon, TheIconInfo);
    {прорисовываем иконку}
    DrawIconEx(Image1.Canvas.Handle,0,0,TheIcon,0,0,0,
    Image1.Canvas.Brush.Handle,DI_DEFAULTSIZE OR DI_NORMAL);
  end
  else
  {...иначе, выдаем информацию о курсоре}
  begin
    {загружаем выбранный системный курсор}
    TheCursor:=LoadCursor(0, CursorTypes[ComboBox1.ItemIndex]);
    {заполняем информационную структуру об этом курсоре}
    GetIconInfo(TheCursor, TheIconInfo);
    {прорисовываем курсор}
    DrawIconEx(Image1.Canvas.Handle,0,0,TheCursor,0,0,0,
    Image1.Canvas.Brush.Handle,DI_DEFAULTSIZE OR DI_NORMAL);
  end;

  {очищаем поле информации Edit}
  Edit1.Clear;
  {заполняем поле Edit информацией о курсоре или иконке}
  if TheIconInfo.fIcon then
      Edit1.Text := 'Это иконка: ' + IntToStr(TheIconInfo.xHotspot)+
                    'x' +            IntToStr(TheIconInfo.yHotspot)
  else
      Edit1.Text := 'Это курсор: ' + IntToStr(TheIconInfo.xHotspot)+
                    'x' +            IntToStr(TheIconInfo.yHotspot);
```

```
  {выведем изображения с масками AND и OR}
  Image2.Picture.Bitmap.Handle:=TheIconInfo.hbmMask;
  Image3.Picture.Bitmap.Handle:=TheIconInfo.hbmColor;
end;
```

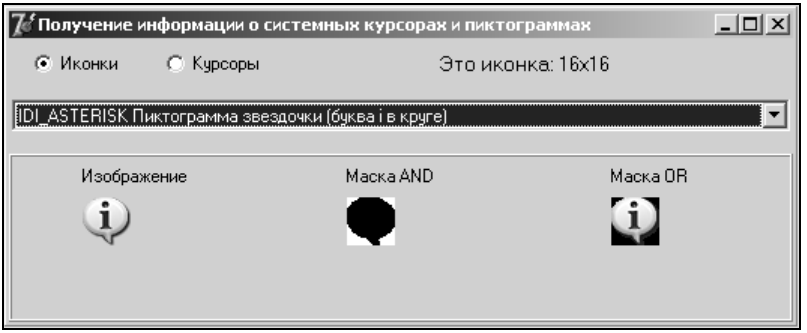


Рис. 6.10. Получение системных курсоров и иконок

В примере, представленном в листинге 6.19, существующее растровое изображение конвертируется в иконку. Возможность конвертирования достигается за счет возможностей функции `ExtractIcon`, которая позволяет считать битовое изображение не только из объектов `Bitmap`, но и из объектов иконок (Icons). Результат выполнения программы представлен на рис. 6.11. Исходный код примера находится на прилагаемом к книге компакт-диске в каталоге Source\Ch06\Ex24.

Листинг 6.19. Конвертирование растрового изображения в иконку и обратно

```
procedure TForm1.FileListBox1Click(Sender: TObject);
begin
  {открываем выбранный файл, как иконку (с автоматическим
конвертированием
  растрового изображения в иконку}
  CurIcon.Handle :=
          ExtractIcon(hInstance, PChar(FileListBox1.FileName), 0);
  Button1.Enabled := TRUE;
  {очищаем PaintBox}
  PaintBox1.Canvas.Brush.Color := clBtnFace;
  PaintBox1.Canvas.FillRect(PaintBox1.ClientRect);
  {если пользователь выбирает конвертирование изображения в иконку,
    то...}
  if RadioButton1.Checked then
  begin
```

```
      {очищаем текущее изображение}
      CurBitmap.Canvas.Brush.Color := clBtnFace;
      CurBitmap.Canvas.FillRect(PaintBox1.ClientRect);
      {прорисовываем иконку}
      DrawIcon(CurBitmap.Canvas.Handle, 0, 0, CurIcon.Handle);
      {отображаем имя растрового изображения}
      PaintBox1.Canvas.Draw((PaintBox1.Width div 2)-16,
                            (PaintBox1.Height div 2)-16, CurBitmap);
  end
  else
     {отображаем иконку}
     DrawIcon(PaintBox1.Canvas.Handle, (PaintBox1.Width div 2)-16,
             (PaintBox1.Height div 2)-16, CurIcon.Handle);
end;

procedure TForm1.RadioButton1Click(Sender: TObject);
begin
 {если пользователь выбирает конвертирование иконки в изображение, то...}
 if Sender=RadioButton1 then
 begin
    {устанавливаем фильтр, чтобы отображались только иконки}
    FileListBox1.Mask := '*.ico';
    {инициализируем соответственно TSavePictureDialog}
    SavePictureDialog1.Filter := 'Bitmaps (*.bmp)|*.bmp';
    SavePictureDialog1.DefaultExt := '*.bmp';
 end
 else
 begin
    {иначе устанавливаем фильтр, чтобы отображались только файлы *.bmp}
    FileListBox1.Mask := '*.bmp';
    {инициализируем соответственно TSavePictureDialog}
    SavePictureDialog1.Filter := 'Icons (*.ico)|*.ico';
    SavePictureDialog1.DefaultExt := '*.ico';
 end;
 {очищаем текущее изображение}
 PaintBox1.Canvas.Brush.Color := clBtnFace;
 PaintBox1.Canvas.FillRect(PaintBox1.ClientRect);
 Button1.Enabled := FALSE;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  {создаем объект иконки для хранения конвертированного растрового
   изображения}
  CurIcon := TIcon.Create;
```

```
  {создаем объект растрового изображения для хранения
   конвертированной иконки}
  CurBitmap := TBitmap.Create;
  CurBitmap.Width := GetSystemMetrics(SM_CXICON);
  CurBitmap.Height := GetSystemMetrics(SM_CYICON);
  {инициализируем соответственно TSavePictureDialog}
  SavePictureDialog1.Filter := 'Bitmaps (*.bmp)|*.bmp';
  SavePictureDialog1.DefaultExt := '*.bmp';
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  {удаляем последнее выбранное имя файла}
  SavePictureDialog1.FileName := '';
  {сохраняем рисунок в файл}
  if SavePictureDialog1.Execute then
  if RadioButton1.Checked then
     {если выбран этот переключатель, то сохраняем как растр...}
     CurBitmap.SaveToFile(SavePictureDialog1.FileName)
  else
     {...иначе, как иконку}
     CurIcon.SaveToFile(SavePictureDialog1.FileName);
end;
```

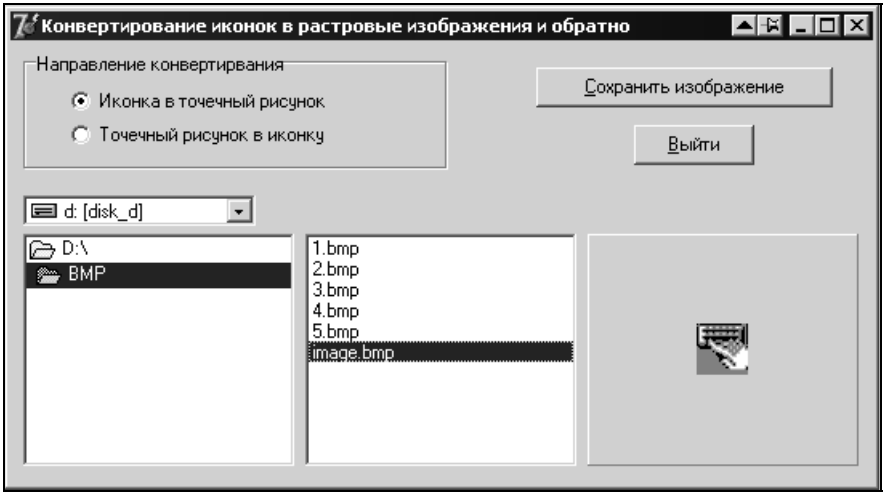


Рис. 6.11. Конвертирование растрового изображения в иконку и обратно

Заключение

В данной главе мы постарались раскрыть вам дверь в огромный мир WinAPI. Конечно, привередливый читатель скажет, что данной информации недостаточно для полного освещения WinAPI. На что авторы, склоня голову, согласятся, да, недостаточно, но и глава была написана лишь для введения в мир WinAPI, и она ни в коей мере не претендует на роль опытного гида в данном мире. Если вы хотите получить полную техническую информацию, изучите подробное описание (help) или многочисленные справочники по функциям, структурам и константам WinAPI. Однако данные справочники занимают тысячи страниц, и, согласитесь, раздувать главу до таких размеров было бы нецелесообразно. Глава, прежде всего, ориентирована на начинающих программистов, вступающих на трудную, но интересную дорогу мира WinAPI.

ПРИЛОЖЕНИЕ

Описание компакт-диска

Прилагаемый компакт-диск содержит заголовочные файлы соответствующих API, справочную информацию, а также проекты Delphi, приведенные в книге в качестве примеров. Каждый пример содержит исполняемый файл, что позволяет запустить приложение непосредственно с компакт-диска без каких-либо предварительных действий.

На компакт-диске находятся следующие материалы:

Каталог	Краткое описание
Doc	Различная справочная информация по API
Install	Заголовочные файлы соответствующих интерфейсов прикладного программирования для Delphi — GDI+, TAPI, VFW
Install\GDI+ SDK	Инсталляционный пакет для развертывания GDI+ на компьютере пользователя
Source	Демонстрационные примеры, прилагаемые к книге, в соответствии с номерами глав, где о них идет речь

Предметный указатель

A

ActivateKeyboardLayout 334
AddAtom 337, 338
Alpha-канал 251
AVIFileCreateStream 189
AVIFileExit 165
AVIFileGet//Stream 170
AVIFileInfo 167
AVIFileInit 165
AVIFileOpen 165
AVIFileRelease 166
AVIMakeCompressedStream 192
AVISave 199
AVISaveOptions 191
AVISaveOptionsFree 192
AVIStreamGetClose 185
AVIStreamGetFrame 185
AVIStreamGetFrameOpen 185
AVIStreamInfo 171
AVIStreamLength 183
AVIStreamReadFormat 177
AVIStreamRelease 170
AVIStreamSampleToTime 183
AVIStreamSetFormat 192
AVIStreamStart 184
AVIStreamWrite 193

B

BITMAPINFOHEADER 177
BringWindowToTop 303

C

Callback-функции 299
ClipCursor 331
CloseClipboard 343
CloseHandle 315, 325
CloseWindow 306
CLSID кодера изображения 240
CoordinateSpace 270
CopyIcon 357
CreateFile 314, 318
CreateIoCompletionPort 46

D

DeferWindowPos 303
DefWindowProc 294
DeleteAtom 337, 338
DispatchMessage 294
DragDetect 336
DrawIcon 357
DrawImage 227
DrawString 254, 265
Dual EMF+ 228

E

EMF+ 228
EmptyClipboard 343
EncoderParameter 242, 243
EncoderValue 242, 243
EncoderValueFlush 247

EndDoc 282
EndPage 282
EnumWidows 311
ExpandEnvironmentStrings 350
Extended MAPI 11
ExtractIcon 359

F

FillClosedCurve 224
FillEllipse 224
FillPath 224
FillPie 224
FillPolygon 224
FillRectangle 224
FillRectangles 224
FlushFileBuffers 324
FontStyle 255
Four Character Code 174
FourCC 174
FrameDimensionPage 246
FrameDimensionTime 246

G

GDI+ 206
◊ константы и структуры 222
◊ координатная система 267
◊ методы рисования графических примитивов 223
◊ перечисления 221
GetCellAscent 262
GetCellDescent 262
GetClipboardData 343
GetCommandLine 350
GetDeviceCap 285
GetEmHeight 262
GetEncoderClsid 241
GetEncoderParameterList 242
GetEncoderParameterListSize 242
GetFrame 185
GetFrameCount 246
GetIconInfo 357
GetImageDecoders 239
GetImageEncoders 239
GetImageEncodersSize 239
GetLineSpacing 262
GetMessagePos 294
GetMessageTime 294
GetNextWindow 303
GetPixel 253
GetProp 307
GetPropertyItem 249
GetQueuedCompletionStatus 46
GetThumbnailImage 236
GetTopWindow 303
GetWindowText 311, 313
GlobalAddAtom 337, 338
GlobalDeleteAtom 337, 338

H

Handle 292, 293
Handoff 97
HKEY 292
HMENU 292
HWND 292

I

ImageCodecInfo 239
InitAtomTable 337
InterpolationMode 233
IsCharAlphaNumeric 342
IsIconic 306
IsVisible 282

L

lineAccept 101, 105
LINEADDRESSCAPS 47
lineAnswer 101
LineCallback 59
LINECALLINFO 109
LINECALLPARAMFLAGS_ 84
LINECALLPARAMS 77, 79
LINECALLSTATUS 109, 116
lineClose 49
lineConfigDialog 41
lineConfigDialogEdit 41
lineDeallocateCall 102
LINEDEVCAPS 47
lineDial 77, 78
lineDrop 78, 102
lineGetAddressCaps 41, 47, 48
lineGetCallInfo 100, 108
lineGetCallStatus 109, 115
lineGetDevCaps 41, 47
lineGetDevConfig 43
lineGetID 43, 50
lineGetMessage 66
lineGetStatusMessages 69
lineHandoff 97

lineInitialize 40
lineInitializeEx 40
LINEINITIALIZEEXPARAMS 41, 67
lineMakeCall 75, 79
LINEMESSAGE 68
lineMonitorMedia 98
lineNegotiateAPIVersion 47
lineNegotiateExtVersion 47
lineOpen 49
lineSetCallPrivilege 70
lineSetDevConfig 42
lineSetMediaMode 97
lineSetStatusMessages 70
lineShutdown 44
lineTranslateAddress 86
lineTranslateDialog 89
LINETRANSLATEOUTPUT 87
LoadKeyboardLayout 334
LockFile 324

M

MakeColor 251
MakeIntAtom 337
MAPI 11
MAPIAddress 23
MAPIDeleteMail 33
MAPIDetails 26
MAPIFindNext 28
MAPIFreeBuffer 27
MAPILogOff 16
MAPILogon 15
MAPIReadMail 29
MAPIResolveName 26
MAPISaveMail 32
MAPISendDocuments 19
MAPISendMail 18
MCI_GETDEVCAPS 124
MciGetCreatorTask 129
MciGetDeviceID 128
MciGetErrorString 120, 129
MciGetYieldProc 129
MciSendCommand 120
MciSendString 119
MciSetYieldProc 129
MeasureString 265
Media Control Interface 118
MM_MCINOTIFY 131
MM_MCISIGNAL 131
MoveWindow 304
Multiply 276
MultiplyTransform 277

O

OpenClipboard 343
OpenIcon 306

P

PAVIFILE 166
PBX 40
Plain Old Telephone Service 38
PostQueuedCompletionStatus 45
POTS 38
PropertyItem 249
Public Branch Exchange 40

R

ReadFile 322, 323
Recoloring 252
RemoveProp 307
ResetClip 281
ResetTransform 277
Rotate 276
RotateAt 276
RotateTransform 277, 279

S

Save 243, 247
SaveAdd 247
Scale 276
ScaleTransform 277, 278
SelectActiveFrame 246
SetAlignment 258
SetCaretBlinkTime 337
SetClip 281
SetClipboardData 343
SetColorMatrix 252
SetFilePointer 314, 323
SetFormatFlags 260
SetInterpolationMode 233
SetLineAlignment 258
SetPageScale 269
SetPageUnit 268
SetPixel 253
SetProp 307
SetPropertyItem 249
SetSmoothingMode 233
SetTabStops 259
SetTextRenderingHint 263
SetTransform 277

SetValue 253
SetWindowPos 303
Shear 276
Simple MAPI 11
SmoothingMode 234
StartDoc 282
StartPage 282
StringAlignment 258
StringFormat 255, 258
StringFormatFlags 260
SystemParametersInfo 353

T

TAPI 36
◊ версионность 46
◊ дескриптор 44
tapiGetLocationInfo 75
tapiRequestMakeCall 74
TAVICompressOptions 189, 192
TAVIFileInfo 167
TAVIStreamInfo 192
TColorMatrix 252
TDocInfo 283
Telephony Application Programming Interface 36
Telephony Service Provider Interface 37
TextRenderHint 263
TextRenderingHintAntiAlias 264
TextRenderingHintClearTypeGridFit 264
THandle 292
Thumbnail 226
TMAPIFileDesc 19
TMAPIMessage 16
TMAPIRecipDesc 19
TMessage 295
TPrinter 282
TransformPoints 269
Translate 276
TranslateTransform 267, 277, 279

U

Unit 255
UnloadKeyboardLayout 334
UnlockFile 324

V

VarString 42
VFW 164
Video for Windows 164

W

WAVEFORMATEX 177
WindowFromPoint 313
Windows Metafiles 228
WMF 228
WriteFile 322

Z

Z order 302
Z-порядок 302

А

Ассистент телефонии 73

Б

Буфер обмена 343

Г

Глобальная таблица атомов 337
Глобальное преобразование 277
Графический объект "регион" 280

Д

Дескриптор 292
Директива:
◊ message 294
◊ stdcall 298

И

Идентификатор устройства линии 44
Импортирование функций Windows 298
Интерфейс:
◊ командных строк MCI 118
◊ команд-сообщений MCI 119

К

Классификация MCI-команд 123
Классы:
◊ AdjustableArrowCap 212
◊ Bitmap 213
◊ BitmapData 213
◊ Brush 213
◊ CachedBitmap 213
◊ GDI+ 212
◊ CharasterRange 213
◊ Color 214
◊ CustomLineCap 214
◊ EncoderParametr 214
◊ EncoderParametrs 215
◊ Font 215
◊ FontCollection 215
◊ FontFamily 215
◊ GDIPlusBase 215
◊ Graphics 215
◊ GraphicsPath 216
◊ GraphicsPathIterator 216
◊ HatchBrush 216
◊ Image 217
◊ ImageAttributes 217
◊ ImageCodecInfo 217
◊ InstalledFontCollection 217
◊ LinearGradientBrush 218
◊ Matrix 218
◊ Metafile 218
◊ MetafileHeader 218
◊ PathData 218
◊ PathGradientBrush 218
◊ Pen 219
◊ Point 219
◊ PointF 219
◊ PrivateFontCollection 219
◊ PropertyItem 219
◊ Rect 220
◊ RectF 220
◊ Region 220
◊ Size 220
◊ SizeF 220
◊ SolidBrush 220
◊ StringFormat 220
◊ TextureBrush 221
Коды ошибок:
◊ MCI 157
◊ VFW 203, 204, 205
Композитное (составное) преобразование 276
Константы и идентификаторы устройств MCI 122, 123

М

Макросы форматов позиционирования MCI 129, 130
Мини-просмотр 226
Мировое преобразование 277

С

Сообщения:
◊ Windows 293
◊ линии TAPI 60
Структуры данных MCI 133

Т

Таб-стоп 258
Таблица атомов:
◊ глобальная337
◊ локальная 337

У

Устройства линии 40

Ф

Флаги общие для MCI-команд 133
Форматы:
◊ данных буфера обмена 344
◊ номеров телефонов в TAPI 71
Фрейм 246
Функции обратного вызова 299

Ц

Цветовые вычисления 252