CLUB Computer

ПРОГРАММИРОВАНИЕ ЗВУКА для DOS и WINDOWS

Nathan Gurewich
Ori Gurewich

PROGRAMMING SOUND for DOS and WINDOWS

SAMS
PUBLISHING

Натан ГУРЕВИЧ
Ори ГУРЕВИЧ

ПРОГРАММИРОВАНИЕ ЗВУКА для DOS и WINDOWS

БИНОМ
Москва 1995

УДК 681.3.06

Издание выпущено при содействии Комитета РФ по печати

Натан Гуревич, Ори Гуревич. Программирование звука для DOS и Windows: Пер. с англ.— М.: БИНОМ; НАУЧНАЯ КНИГА, 1995. — 400 с.: ил.
ISBN 0-672-30309-4
ISBN 5-7503-0074-9
ISBN 5-7671-0002-0

Данная книга научит пользователя создавать "звучащие" программы для персонального компьютера. Вы узнаете, как включать в свои программы звуковые файлы, воспроизводить звуковые файлы из ваших программ, определять, установлена ли в персональном компьютере звуковая плата, направлять звуковой файл для воспроизведения как на встроенный динамик персонального компьютера, так и на звуковую плату. Книга содержит множество примеров текстов программ для Windows и DOS, на основе которых вы сможете создавать самые разнообразные демонстрации, включающие как воспроизведение звука, так и анимацию.

Совместное издание ТОО "БИНОМ" и АОЗТ "Научная книга"

Autorized translation from the English language edition.

ISBN 0-672-30309-4
ISBN 5-7503-0074-9
ISBN 5-7671-0002-0

Благодарности

Мы хотим поблагодарить Стейси Хайквит, ответственного редактора издательства Sams Publishing, за поддержку этого проекта и особенно за предложения и рекомендации, которые она высказывала в процессе создания книги.

Мы выражаем благодарность Джуди Брунетти, редактору этой книги, за превосходное редактирование рукописи, за множество советов и консультаций, а также за те дискуссии, которые она вела с нами во время работы над книгой.

Мы хотим поблагодарить также Брюса Грэйвса, технического редактора, который скомпилировал и скомпоновал все программы этой книги, чтобы убедиться, что они действительно компилируются, компонуются и выполняются корректно.

Об авторах

Натан Гуревич — магистр электротехники Колумбийского университета, Нью-Йорк, бакалавр электротехники университета Хофстра, Лонг Айленд, Нью-Йорк. Неоднократно принимал участие в разработке и реализации коммерческих программных продуктов для персональных компьютеров. Является экспертом в области программирования для ПК, оказывает консультационные услуги в области локальных и глобальных вычислительных сетей, разработки баз данных и управления ними.

Ори Гуревич — бакалавр техники университета Стоуни Брук, Стоуни Брук, Нью-Йорк. Работал в качестве старшего инженера-программиста и инженера-консультанта в компаниях, разрабатывающих профессиональные программы для мультимедиа и Windows. Эксперт в области программирования для ПК и сетевых коммуникаций; разработал несколько алгоритмов мультимедиа и звукового сопровождения для ПК.

Глава 1. Программирование звука

Хотите, чтобы персональный компьютер разговаривал и воспроизводил музыку с помощью написанной вами программы? Если да, то эта книга — для вас!

Концепция воспроизведения речи и музыки на персональном компьютере

Персональные компьютеры с каждым днем становятся все менее дорогостоящими и вместе с тем все более мощными и популярными. Они используются не только практически во всех сферах бизнеса, но и в наших домах. Неотъемлемым компонентом персонального компьютера является управляющее им программное обеспечение, которое сейчас тоже становится более мощным, более конкурентоспособным и более дешевым. К примеру, еще совсем недавно существовало только несколько хороших текстовых процессоров. Сегодня не только значительно возросло их число, но и во много раз уменьшилась цена каждой из подобных программ.

Поскольку быстродействие и мощность персональных компьютеров увеличились, пользователи вправе надеяться, что программное обеспечение будет позволять решать одновременно большее количество задач и делать это более быстро, с использованием всех имеющихся возможностей. Оболочка MS Windows — наглядный пример эволюции аппаратного и программного обеспечения персональных компьютеров. Она лучше всего работает на мощных персональных компьютерах, которые имеют высокоскоростные процессоры Intel 80386 или Intel 80486, большой объем оперативной памяти, жесткий диск достаточно большого объема и VGA или другой монитор с высокой разрешающей способностью.

Прогресс в области развития средств аппаратного обеспечения ставит новые задачи перед производителями программных продуктов. Еще несколько лет назад работа с персональным компьютером требовала от пользователя освоения сложных технических приемов управления мощным программным обеспечением, и именно пользователей принято было обвинять в неумении правильно обращаться с неудобными в работе программами. Сегодня же необходимость в изучении таких технических приемов отпала сама собой, поскольку программисты получили возможность создавать удобные программные продукты, доступные для широкого круга пользователей.

С появлением Windows границы требований к квалификации конечного пользователя определились достаточно четко. Прежде всего квалифицированный пользователь должен знать, как работает графическая оболочка Windows. Предполагается, что он умеет запускать программу, щелкнув мышью на ее пиктограмме; выполнять некоторые программы; копировать, вырезать и вставлять текстовые и графические объекты из одной программы в другую; входить в меню программы и выполнять другие действия общего характера с помощью Windows.

Определились и требования к программистам: создавать такие программы, с которыми пользователь, обученный выполнять указанные основные операции, сможет работать без каких-либо затруднений.

Программы, привлекательные для пользователя

Ваша программа должна быть не только простой в эксплуатации, но и удобной в применении, а значит — привлекательной для пользователя, для его зрительного и слухового восприятия. Чтобы этого достичь, для функций отображения следует применить

графический режим, по возможности в цвете. Кроме того, полезно использовать в программе анимацию (например, для создания в приложении под Windows иллюзии нажатия кнопки при щелчке на ней).

И наконец, целесообразно "научить" программу разговаривать и воспроизводить настоящую музыку.

Несколько лет назад почти все программы использовали встроенный динамик персонального компьютера для подачи звуковых сигналов, извещавших пользователя о том, что выполнены некоторые условия, или о том, что во время работы программы возникла какая-либо требующая вмешательства человека ситуация. В настоящее время качественно написанная программа может обращаться к пользователю не посредством раздражающих сигналов, а человеческим голосом.

Существуют два способа воспроизведения человеческого голоса на персональном компьютере: с помощью звуковой платы и с использованием встроенного динамика, который имеется в каждом персональном компьютере.

Концепция компьютерного воспроизведения речи с применением дополнительного аппаратного обеспечения и без него

Персональный компьютер (ПК) — это вычислительная машина, с помощью которой можно решать практически все мыслимые задачи. Однако существует одна задача, на которую персональный компьютер не был первоначально ориентирован: это воспроизведение настоящей человеческой речи и настоящей музыки.

Для того чтобы восполнить это "упущение", была разработана звуковая плата — периферийное устройство, которое вставляется в один из разъемов материнской платы персонального компьютера. Звуковая плата имеет разъемы для подключения к ней внешнего микрофона и внешних динамиков. Ваша программа может посылать звуковой плате команды, заставляющие ее воспроизводить звуковые файлы.

Работа со звуковой платой предполагает использование более сложного программного обеспечения и требует подключения дополнительных устройств — микрофона и внешних динамиков.

Реалии сегодняшнего дня таковы, что не все персональные компьютеры оснащены звуковыми платами. Поэтому в данной книге рассматриваются программы на языке C, исполняемые под DOS и Windows, которые рассчитаны на воспроизведение музыки и речи (т.е. речевых и музыкальных файлов) как с помощью звуковой платы, так и через встроенный динамик. Такие речевые файлы (воспроизводимые на персональном компьютере как с помощью звуковой платы, так и с помощью динамика) содержат запись настоящей (а не синтезированной) человеческой речи, а музыкальные файлы — записи настоящей музыки.

Кроме этого, вы научитесь определять программным путем, установлена или нет в вашем компьютере звуковая плата. Обнаружив отсутствие последней, ваша программа воспроизведет звуковой файл через встроенный динамик ПК. Этот маленький динамик, который имеется во всех персональных компьютерах, может воспроизводить настоящую человеческую речь и настоящую музыку. Концепция воспроизведения звуковых файлов на ПК схематично изображена на рис. 1.1.

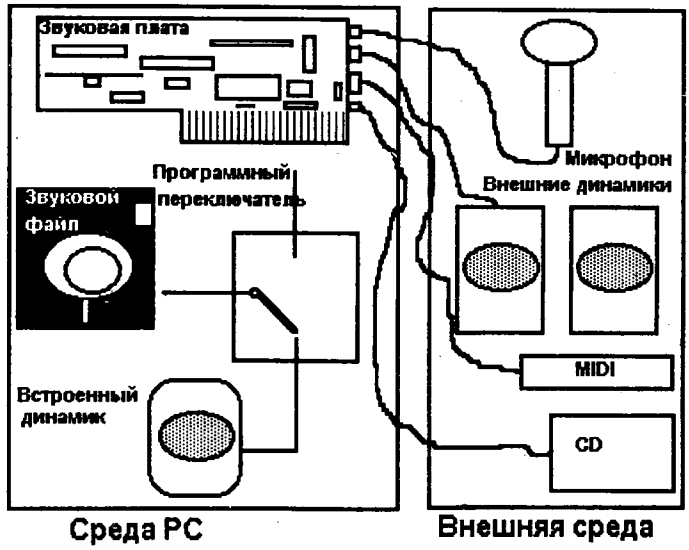


Рис. 1.1. Концепция воспроизведения звуковых файлов на персональном компьютере

Возможно, вам уже приходилось использовать встроенный динамик персонального компьютера для воспроизведения в процессе выполнения программы некоторых примитивных сигналов. Но ведь этого крайне мало! Наша книга научит вас заменять все эти надоедливые противные гудки осмысленными фразами, и после того как ваша программа однажды "заговорит", с "немым" программным обеспечением будет покончено раз и навсегда.

Инсталляция прилагаемой к книге дискеты

Для создания на языке C программ, позволяющих записывать и воспроизводить звуковые файлы, необходимо определенное программное обеспечение. Дискета, входящая в комплект этой книги, содержит библиотеки функций C, которые вы сможете использовать в своих программах. Для этого последние должны быть написаны на C, а затем откомпилированы с помощью компилятора Microsoft C или Borland C и скомпонованы.

TSEngine

Библиотеки функций C, включенные в предлагаемую дискету, — это *обычные* функции C, которые могут вызываться из ваших программ на языке C, исполняемых под DOS и под Windows. Они позволяют очень легко организовать звуковое сопровождение программ. Этот мощный набор функций C именуется в данной книге TSEngine.

Дискета, поставляемая в комплекте с данной книгой

Дискета, входящая в комплект данной книги, содержит звуковые файлы, а также исходные тексты (на языке C):

- программ,
- утилит,

- функций библиотеки C для компилятора Microsoft C, позволяющих создавать приложения под Windows,
- функций библиотеки C для компилятора Borland C, позволяющих создавать приложения под Windows,
- других утилит.

Это программное обеспечение представляет собой сокращенную версию библиотеки TS Sound Plus фирмы TegoSoft Inc. Используя его, вы сможете самостоятельно компилировать, компоновать и выполнять все программы, приведенные в книге, а также создавать собственные простые программы.

Инсталляция дискеты, входящей в комплект с данной книгой

Программное обеспечение, поставляемое вместе с этой книгой, хранится в упакованном формате, и вы не сможете использовать его без предварительной инсталляции на жесткий диск компьютера.

Для инсталляции программного обеспечения на жесткий диск своего персонального компьютера выполните перечисленные ниже действия.

- После приглашения DOS задайте в качестве устройства по умолчанию дисковод, содержащий инсталляционную дискету. Например, если дискета находится в дисководе A:, то введите **A:** и нажмите клавишу [Enter].
- Затем введите **INSTALL** и вновь нажмите клавишу [Enter].

Это приведет к созданию на диске C: директории с именем SPSDK и инсталляции всех файлов в эту директорию. Для инсталляции программного обеспечения, поставляемого с книгой, вам потребуется минимум 4,6 Мбайт свободного пространства на жестком диске компьютера.

Примечание:

Файлы будут инсталлированы на диске C:. В некоторых программах, использующих указанные файлы, логическое имя задается явно, поэтому в случае, если вы инсталлировали программное обеспечение на другой диск, следует модифицировать соответствующие ссылки на файлы в текстах программ.

В процессе инсталляции файлов на жесткий диск будут созданы некоторые поддиректории. Обязательно прочтите в поддиректории LICENSE информацию о лицензии на данное программное обеспечение.

Тексты программ, приведенные в книге, соответствуют содержимому поставляемой вместе с книгой дискеты. Однако возможны и некоторые расхождения. Это объясняется тем, что в последний момент в содержимое дискеты могли быть внесены какие-то изменения (например, включены другие пиктограммы программ). Таким образом, на дискете могут находиться более свежие (откомпилированные, скомпонованные и проверенные) версии приведенных в книге программ.

Как уже отмечалось, вы можете компилировать и компоновать все примеры, приведенные в книге, самостоятельно. Но можно поступить и иначе: воспользоваться уже откомпилированными и скомпонованными программами под Windows, записанными на дискете. Это позволит вам выполнять, а также *слушать* эти программы, не тратя времени на их отладку, и, таким образом, быстрее усвоить изложенный в книге материал. В следующей главе вы узнаете, как выполнять упомянутые программы.

Примеры программ описаны в доступной для понимания форме, последовательно, шаг за шагом. Так что — позвольте пожелать вам приятно провести время!

Глава 2. Техника программирования звука и звуковые библиотеки

При изучении программирования звукового сопровождения возникают два основных вопроса.

1. Как заставить персональный компьютер воспроизводить звуковые файлы через встроенный динамик или с помощью установленной звуковой платы?
2. Как лучше использовать возможности персонального компьютера воспроизводить речь и музыку для создания мощных, впечатляющих программ?

Как заставить персональный компьютер воспроизводить звуковые файлы

Данная книга научит вас воспроизводить с помощью персонального компьютера различные звуки. Для этого необходимо уметь:

- включать в вашу программу звуковые файлы;
- воспроизводить звуковые файлы из вашей программы;
- определять, установлена ли в персональном компьютере звуковая плата;
- направлять звуковой файл для воспроизведения как на встроенный динамик персонального компьютера, так и на звуковую плату.

Все эти задачи выполняются и под Windows, и под DOS.

Умение решать указанные задачи сводится к умению использовать соответствующие функции С. В книге вы встретите множество примеров программ и сможете шаг за шагом проследить процесс их создания.

Как лучше использовать звуковое сопровождение вашей программы

После того как вы научитесь воспроизводить различные звуки из вашей программы и будете в состоянии заставить персональный компьютер звучать, нужно подумать о том, как лучше использовать это новое измерение в ваших программах. Здесь вы можете применить все свои технические таланты и изобретательность. Программирование звука может включаться в программы всех типов: "серьезные" деловые программы, текстовые процессоры, утилиты, анимационные программы, демонстрационные программы и т.п.

Когда вы изучите примеры программ, приведенные в этой книге, вы сможете самостоятельно управлять воспроизведением звука. Вы ознакомитесь с процессами компиляции, компоновки и выполнения этих примеров. Дискета, поставляемая с книгой, содержит исходные тексты всех приведенных в книге программ.

Рассмотрим некоторые из них. Мы рекомендуем выполнять программы перед тем, как приступать к изучению их кода. Этот подход позволит *услышать* и *увидеть* то, что вы будете изучать (и, возможно, впоследствии приобретете).

Оболочка Windows

Оболочка Windows с каждым днем становится все более популярной. В настоящее время почти все производители персональных компьютеров поставляют свои ПК с установленным драйвером мыши и оболочкой Windows.

Создание новой группы программ в Диспетчере Программ

Если на вашем персональном компьютере установлена оболочка Windows, то для создания в Диспетчере Программ Windows новой группы программ, содержащей программы, приведенные в этой книге, выполните описанные ниже действия. Для создания новой группы программ можно использовать два следующих метода.

Метод 1

Вначале посмотрите на рис. 2.1: в Диспетчере Программ Windows содержатся различные группы программ — Главная, Реквизиты и т.п. Сейчас мы добавим в Диспетчер Программ новую группу, которая содержит программы, приведенные в этой книге.

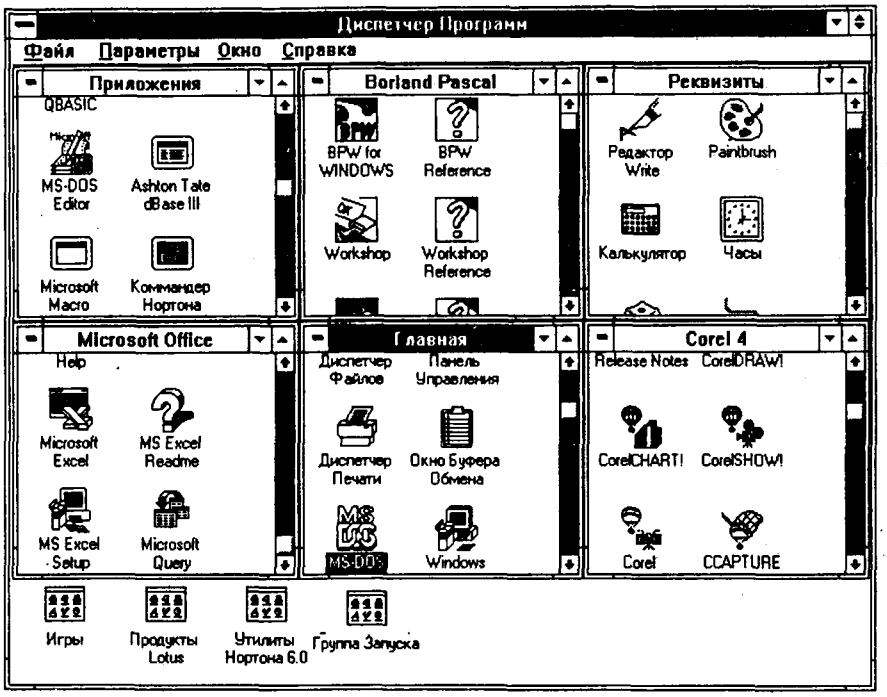


Рис. 2.1. Различные группы программ, созданные в Диспетчере Программ Windows

- Выберите меню **Файл** Диспетчера Программ. Появится меню **Файл**, изображенное на рис. 2.2.
- Выберите из меню **Файл** команду **Создать**. Появится диалоговое окно **Новый Программный Объект**, изображенное на рис. 2.3. Это диалоговое окно позволяет создавать как новую группу программ, так и новый программный элемент. Наша цель — создать новую группу программ, поэтому следует выполнить щелчок на селекторной кнопке **Группа Программ** (см. рис. 2.3).

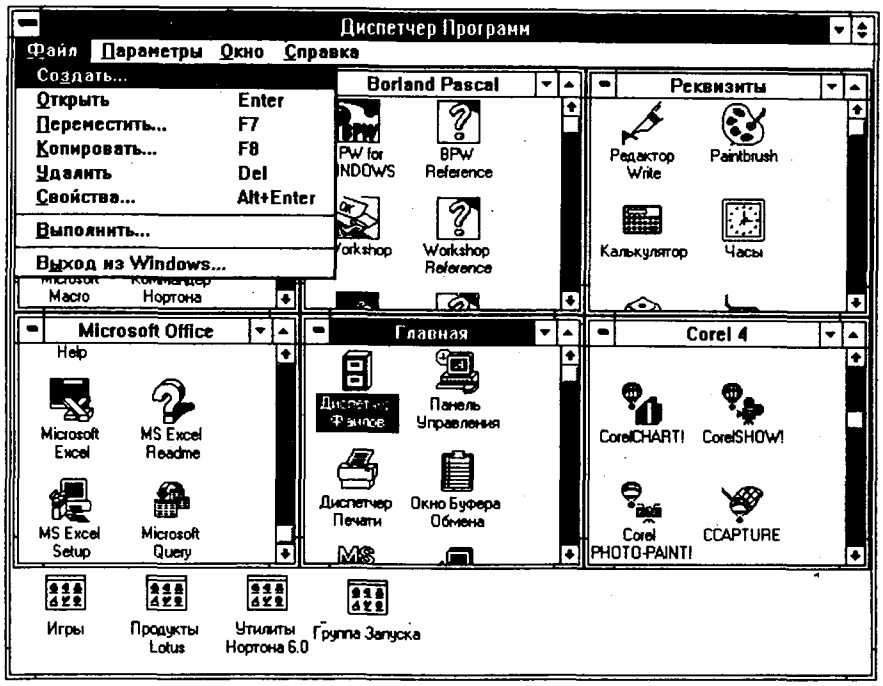


Рис 2.2. Меню Файл Диспетчера Программ Windows

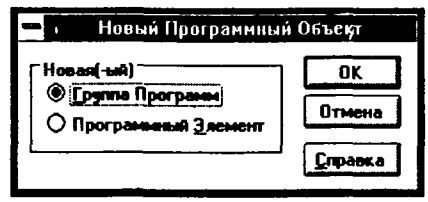


Рис. 2.3. Диалоговое окно Новый Программный Объект

➤ После того как вы нажмете командную кнопку **OK**, появится диалоговое окно **Свойства Группы Программ**. Введите в поле **Описание** диалогового окна **Свойства Группы Программ** (см. рис. 2.4) надпись **Книга Гуревичей**. Поле **Файл Группы** можете оставить пустым.

После нажатия командной кнопки **OK** появится новая группа программ (см. рис. 2.5). Видно что, в этой группе пока нет пиктограмм программ. Сейчас мы добавим несколько пиктограмм, каждая из которых соответствует одной из приведенных в книге программ.

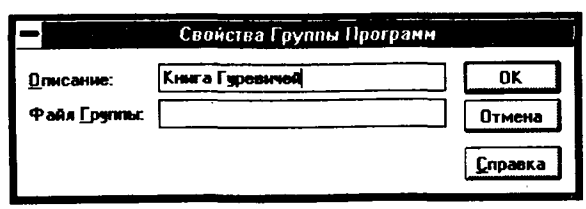


Рис. 2.4. Диалоговое окно Свойства Группы Программ

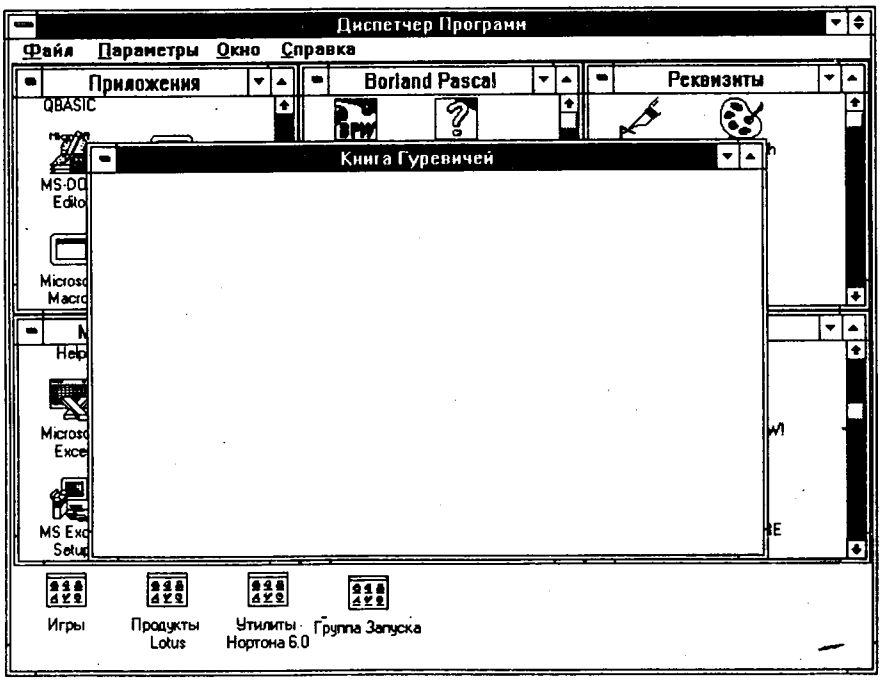


Рис. 2.5. Группа программ Книга Гуревичей

Теперь добавим пиктограммы в группу **Книга Гуревичей**.

- При активном окне **Книга Гуревичей** (заголовок окна **Книга Гуревичей** выделен другим цветом) выберите меню **Файл** Диспетчера Программ. Появляется меню **Файл**, изображенное на рис. 2.2.
- Выберите в этом меню команду **Создать**. Появится диалоговое окно **Новый Программный Объект**, изображенное на рис. 2.3.

  Поскольку мы добавляем новый программный элемент (а не новую группу программ), то нужно нажать селекторную кнопку **Программный Элемент** (а не селекторную кнопку **Группа Программ**).
- В диалоговом окне **Новый Программный Объект** нажмите командную кнопку **OK**. Появляется диалоговое окно **Свойства Программного Элемента**, изображенное на рис. 2.6. Первой добавим пиктограмму программы Dog.
- В поле **Описание** диалогового окна **Свойства Программного Элемента** запишите **Dog**.
- Запишите в поле **Командная Строка** диалогового окна **Свойства Программного Элемента** следующую строку:

```
c:\spSDK\Samp4Win\Dog.exe
```

  В качестве альтернативного способа заполнения поля **Командная Строка** можно использовать командную кнопку **Пролистать**. Как это сделать, описано ниже.
- Нажмите командную кнопку **Пролистать** и выберите директорию c:\spSDK\Samp4Win\.
- Выберите файл DOG.EXE.

После нажатия кнопки **OK** поле **Командная Строка** должно содержать следующую командную строку:

```
C:\SPSDK\SAMP4WIN\DOG.EXE
```

Поле **Рабочий Каталог** диалогового окна **Свойства Программного Элемента** можно оставить пустым.

После того как вы нажмете командную кнопку **OK** диалогового окна **Свойства Программного Элемента**, в группу программ **Книга Гуревичей** добавится новая пиктограмма, как показано на рис. 2.7. Теперь двойным щелчком мыши на пиктограмме **Dog** можно запускать программу Dog.

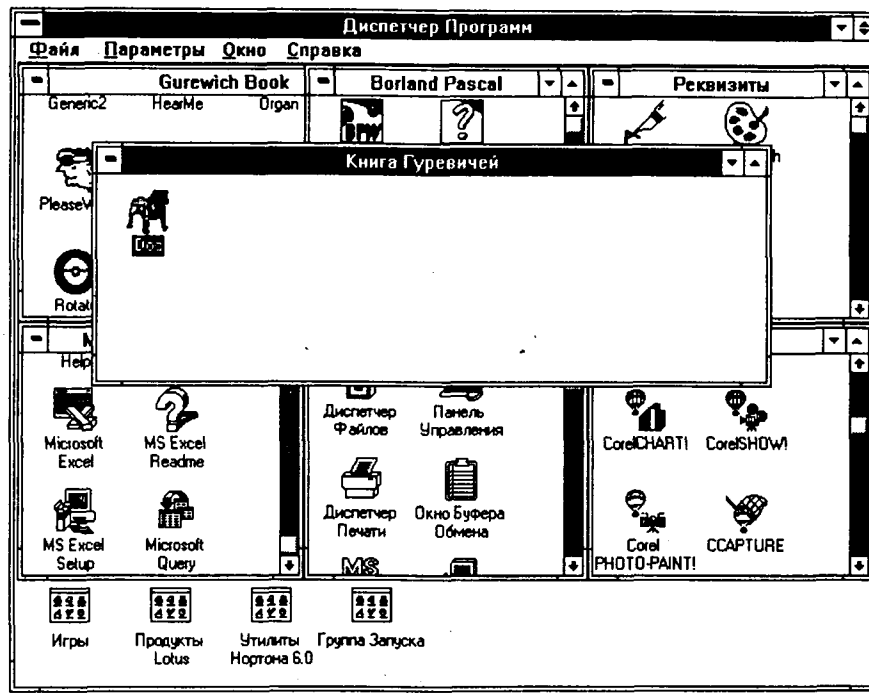


Рис. 2.7. В группу Книга Гуревичей добавлена пиктограмма Dog

Остальные пиктограммы добавляются в группу программ **Книга Гуревичей** путем простого повторения предыдущих действий. Добавьте пиктограммы для каждого EXE-файла, который находится в директории c:\spSDK\Samp4Win\.

Когда вы завершите процесс создания пиктограмм для каждой программы из c:\spSDK\Samp4Win\, группа **Книга Гуревичей** будет содержать пиктограммы, изображенные на рис. 2.8.

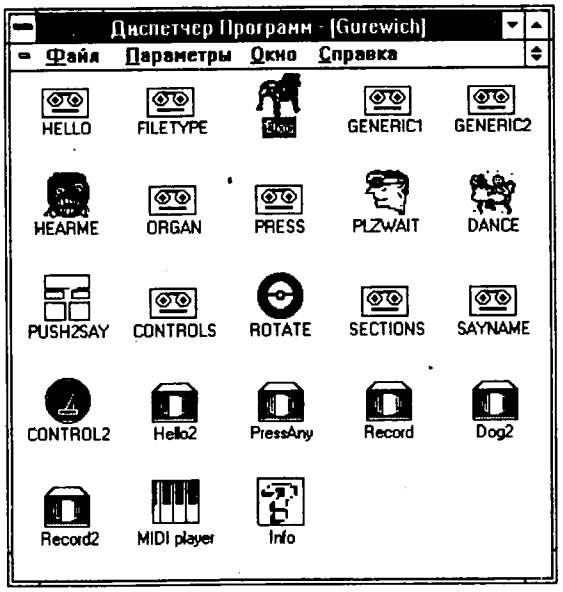


Рис. 2.8. Программы под Windows, приведенные в книге

Метод 2

Альтернативный метод создания новой группы пиктограмм позволяет делать это заметно быстрее.

- Войдите в директорию c:\spSDK\Misc\.
- Скопируйте файл Gurewich.grp в директорию Windows (нажатие клавиши [Enter] по окончании ввода команды обозначено как [Enter]):

```
Copy c:\spSDK\Misc\Gurewich.grp c:\Windows    [Enter]
```

- Из Диспетчера Программ Windows двойным щелчком мыши на пиктограмме **Диспетчер Файлов** запустите программу Диспетчер Файлов (см. рис. 2.9).

Рис. 2.9. Пиктограмма Диспетчер Файлов

Появится окно программы Диспетчер Файлов, изображенное на рис. 2.10.

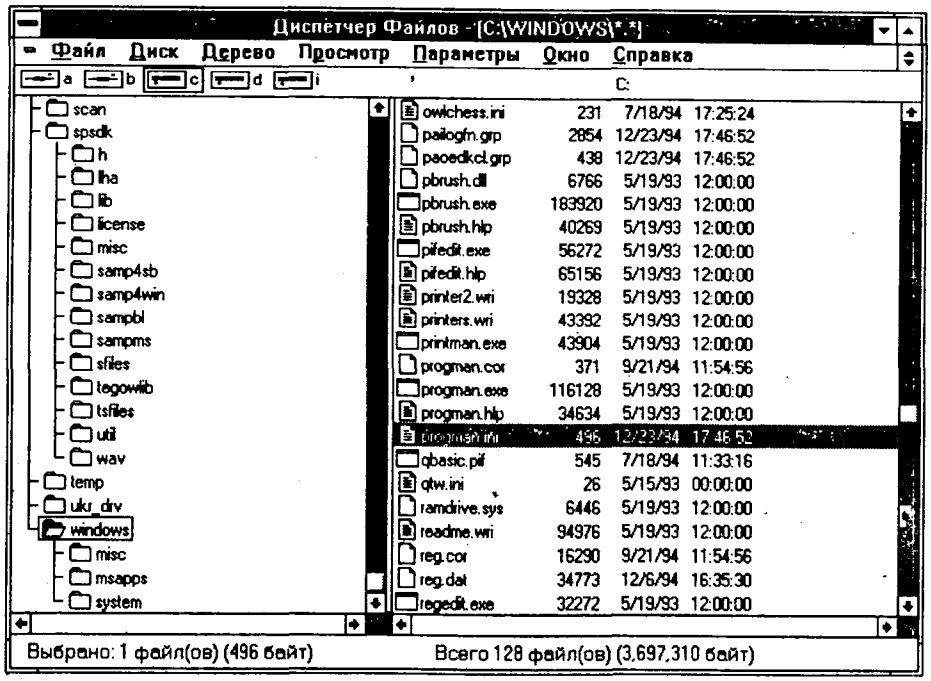


Рис. 2.10. Окно программы Диспетчер Файлов.

➤ Выполните двойной щелчок на файле PROGMAN.INI (этот файл располагается в директории c:\Windows\)

Появится окно программы Блокнот с готовым к редактированию файлом PROGMAN.INI (см. рис. 2.11).

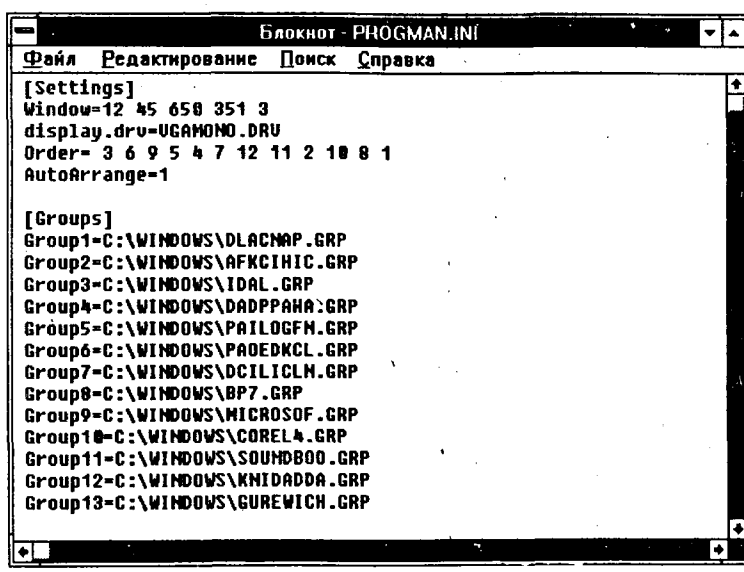


Рис. 2.11. Окно программы Блокнот

- Как видно из рис. 2.11, файл PROGMAN.INI содержит список групп программ. Для того чтобы включить новую группу программ, добавьте строку

  ```
  GroupXX=C:\WINDOWS\GUREWICH.GRP
  ```

  где XX — следующий свободный номер группы.

  На нашем компьютере файл PROGMAN.INI содержал 12 групп программ, следовательно, мы добавили новую группу программ как

  ```
  Group13=C:\WINDOWS\GUREWICH.GRP
  ```

- Чтобы эти изменения вступили в силу, необходимо перезапустить Windows (это можно сделать, выйдя из Windows, а затем опять запустив ее).

Выполнение программ

Вы, вероятно, заметили, что для запуска приведеных в книге программ используются различные пиктограммы. Это стилизованные изображения магнитофонных кассет, внешних динамиков и др.(см. рис. 2.8).

Программы, обозначенные пиктограммами в виде магнитофонной кассеты, могут выполняться на любом персональном компьютере. Вам не потребуется никакого дополнительного аппаратного и программного обеспечения: звуковые файлы воспроизводятся через встроенный динамик.

Программы, обозначенные пиктограммами в виде внешнего динамика, можно выполнять только на компьютере, на котором установлена звуковая плата. Они воспроизводят звуковые файлы с помощью совместимой с Windows звуковой платы.

Попытайтесь выполнить некоторые из этих программ. Например, для выполнения программы Dog нужно дважды щелкнуть на пиктограмме **Dog**, для выполнения программы Dance — соответственно на пиктограмме **Dance** и т.д.

Работа большинства из этих программ сопровождается пояснениями, для понимания принципов работы остальных требуется дополнительная информация. Так, для выполнения программы SayName следует выполнить двойной щелчок на пиктограмме SayName, а затем переключиться в Диспетчер Программ и попытаться выйти из Windows или выполнить программу Paintbrush.

Наша книга научит вас создавать подобные программы.

Структура книги

Материал книги систематизирован следующим образом:

- **Главы 1 и 2** представляют собой обзор методов программирования звукового сопровождения.
- **Главы 3 – 9** содержат сведения о том, как создавать Windows-приложения, которые способны воспроизводить звуки через встроенный динамик ПК без какого-либо дополнительного аппаратного или программного обеспечения.
- **Глава 10** рассказывает о том, как написать автономные звуковые приложения под Windows. Автономное звуковое приложение — это EXE-файл, содержащий звуковые файлы и потому не требующий подключения внешних звуковых файлов.
- **Глава 11** содержит сведения о том, как создавать Windows-приложения, которые воспроизводят и записывают звук с помощью звуковой платы. Звуковая плата обязательно должна быть совместимой с Windows.

- **Глава 12** посвящена вопросам создания DOS-приложений, которые могут воспроизводить звуки через встроенный динамик персонального компьютера без какого-либо дополнительного аппаратного или программного обеспечения.
- **Глава 13** знакомит читателя с тем, как создавать DOS-приложения, которые могут записывать и воспроизводить звуки через звуковую плату Sound Blaster.
- **Приложение** включает информацию о том, как применять звуковую библиотеку DLL. Эта библиотека может использоваться программами, написанными на C, на Visual Basic for Windows и на других языках программирования в среде Windows, которые могут использовать DLL.

Звуковые программы для DOS

Директория c:\spSDK\SampMS\ содержит программы на C, которые необходимо компилировать и компоновать компилятором Microsoft C. В директории c:\spSDK\SampBL\ размещены программы на C, которые должны компилироваться и компоноваться компилятором Borland C. Эти программы не были записаны на дискету, поставляемую с книгой, из-за ограниченного объема памяти, поэтому в главе 12 описано, как производить их компиляцию и компоновку. Чтобы читатель смог получить представление об этих программах, содержимое дискеты включает две скомпонованные демонстрационные DOS-программы (в директории c:\spSDK\Util\), которые могут воспроизводить звуковые S-файлы, содержащиеся в директории c:\spSDK\Sfiles.

Программу GR1.EXE можно выполнить следующим образом:

- Войдите в директорию c:\spSDK\Util\.
- Для воспроизведения файла Day.s введите после приглашения DOS:

```
GR1 c:\spSDK\Sfiles\Day.s    [Enter]
```

Аналогичным образом вы можете воспроизвести и другие S-файлы, которые располагаются в директории c:\spSDK\Sfiles\.

Программа GR2.EXE выполняется следующим образом:

- Войдите в директорию c:\spSDK\Util\.
- Для воспроизведения файла Day.s введите после приглашения DOS:

```
GR2 c:\spSDK\Sfiles\Day.s    [Enter]
```

Аналогичным образом воспроизводятся и другие S-файлы, содержащиеся в директории c:\spSDK\Sfiles\.

Приведенные в качестве примеров программы GR1.EXE и GR2.EXE воспроизводят только звуковые S-файлы. Далее вы узнаете, как писать программы, которые воспроизводят и звуковые файлы других типов, например, звуковые файлы с расширениями .TS, а также файлы с расширениями .WAV и .VOC.

Глава 3. Программа Generic1 для Windows

Сейчас мы рассмотрим Windows-программу Generic1.c, которая служит основой для многих последующих программ.

Программы Generic1.c и Generic2.c

Известно, что программы на C, исполняемые под Windows, как правило, имеют больший размер, чем программы на C под DOS. Однако все приложения Windows похожи друг на друга, и во многих книгах, обучающих программированию под Windows, вначале приведена простая шаблонная программа, обычно именуемая Generic.c. Когда вы поняли эту программу и все связанные с ней файлы, вы просто копируете Generic.c и все необходимые для нее файлы. Затем вносите в новые файлы минимальные изменения и создаете новое приложение Windows, причем вводить довольно объемные части программы, общие для всех приложений Windows, уже нет необходимости.

Мы также придерживаемся техники Generic.c. Однако, если ваша программа представляет собой приложение Windows, содержащее звуковое сопровождение, для ее написания можно применить один из двух основных методов разработки таких приложений. А поскольку эти два метода заметно отличаются один от другого, то нам потребуются две программы типа "Generic" — Generic1 и Generic2.

Начнем с Generic1.c, с которой вы, вероятно, уже знакомы. А после того как рассмотрим разработку Windows-приложений, имеющих в своей основе Generic1.c, мы будем писать программы, основанные на Generic2.c.

Программы, основанные на Generic2.c, гораздо мощнее, чем основанные на Generic1.c. Их разработка описана в последующих главах. А сейчас хочется отметить только то, что на основе программы Generic2.c можно писать несколько фантастические, имеющие широкие возможности, многозадачные программы. Например, программа, основанная на Generic2.c, может воспроизводить звуковой файл, а параллельно позволяет переключиться в Диспетчер Программ Windows, выбрать пиктограмму **MS-DOS** (обычно в группе программ **Главная**) и войти в DOS. Находясь в DOS, ПК будет продолжать воспроизводить звуковой файл. Вы сможете также выполнить обычную программу DOS, на фоне выполнения которой звуковой файл все еще будет воспроизводиться. Другой вариант — это переключиться в Word for Windows, Paintbrush или любое другое приложение: когда вы будете работать с этим приложением, ПК в фоновом режиме продолжит воспроизведение звукового файла.

Файлы программы Generic1.c

Программа Generic1 состоит из следующих файлов:

Generic1.h	include-файл
Generic1.c	Текст программы на C
Generic1.rc	Файл ресурсов
Generic1.def	Файл определения модуля
Tape.ico	Пиктограмма

Для компиляции и компоновки Generic1 вам потребуется шестой файл — Generic1.mak:

Generic1.mak	Make-файл программы

Эти файлы доступны в директории c:\spSDK\Samp4Win\. Все файлы программы Generic1 приведены в листингах 3.1–3.5.

Листинг 3.1. Generic1.h

```
/*================================================
 FILE NAME: Generic1.h

 (C) Copyright Gurewich 1992, 1993
 ================================================*/
/*---------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc      ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG );

/*------
 #define
 -------*/
#define IDM_QUIT  1  /* Команда Завершить в меню */
#define IDM_ABOUT 2  /* Команда О программе в меню */

/*--------------------
 Глобальные переменные
 ---------------------*/
char       gszAppName[] = "Generic1" ; /* Имя нашего приложения.*/
HINSTANCE  ghInst;                     /* текущий экземпляр.    */
```

Листинг 3.2. Generic1.c

```
/*============================================================
  ПРОГРАММА: Generic1.c
  ----------
  (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

  ОПИСАНИЕ ПРОГРАММЫ:
  -------------------
  Это программа типа Generic1.
 ============================================================*/

/*--------
 #include
 --------*/
/*-----------------------------------------------
 windows.h требуется во всех приложениях Windows.
 -----------------------------------------------*/
#include <windows.h>

/*-----------------------------------------------
 Требуется для того, чтобы функции sp_, макросы SP
 и определения типа #difine из библиотеки TS Sound
 могли быть использованы в этих приложениях.
 -----------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"
```

```
/*-------------------------------------------------------------
 Определения и прототипы, характерные для этого приложения.
 -------------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\Generic1.h"

/*=================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HWND       hWnd;    /* Указатель на окно нашего приложения.     */
MSG         msg;    /* Сообщение, которое будет обрабатываться */
                    /*  нашим приложением.                      */
WNDCLASS     wc;    /* Класс окна нашего приложения.           */

/*-------------------------------------
 Введем глобальную переменную hInstance.
 -------------------------------------*/
ghInst = hInstance;

/*-----------------------------
 Обновляем структуру класса окна
 и регистрируем класс окна.
 -----------------------------*/
if ( !hPrevInstance )
   {
   /*-------------------------------------
   Условие if выполнено, это самый первый
   запуск этого приложения.
   -------------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfTape" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW      ) ;
   wc.hbrBackground = GetStockObject ( WHITE_BRUSH );
   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = gszAppName ;

   /*----------------
   Регистрируем окно.
   ----------------*/
   RegisterClass ( &wc );
   }/* конец условия if( !hPrevInstance ) */

/*-------------------------------
  Создаем окно нашего приложения.
 -------------------------------*/
```

```
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );
/*-------------------------------------------
Отображаем и обновляем окно нашего приложения.
-------------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );

/*------------
Цикл сообщений.
---------------*/
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage  ( &msg );
      }

return msg.wParam ;

} /* Конец функции. */
/*========================= конец WinMain() ====================*/

/*================
ФУНКЦИЯ: WndProc()
==================*/
/*-----------------------------
ОПИСАНИЕ: Обработка сообщений.
-------------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
{
/*---------------------------------
Локальные и статические переменные.
-----------------------------------*/
HDC          hdc;      /* Необходима для отображения текста. */
PAINTSTRUCT  ps;       /* Необходима для отображения текста. */
RECT         rect;     /* Необходима для отображения текста. */

/*-----------------------------------------
Необходима для диалогового окна О программе.
-----------------------------------------*/
static FARPROC lpfnAboutDlgProc ;
```

```
switch ( message )
        {
        case WM_CREATE:
             /*----------------------------------------
             Обычно здесь вы открываете звуковой сеанс.
             ----------------------------------------*/
             /*........................................*/
             /*... Здесь открываем звуковой сеанс .....*/
             /*........................................*/

             /*--------------------------------------------------
             Получаем lpfnAboutDlgProc диалогового окна О программе.
             --------------------------------------------------*/
             lpfnAboutDlgProc =
             MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);
             return 0;
        case WM_PAINT:
             hdc = BeginPaint ( hWnd, &ps );
             GetClientRect ( hWnd, &rect );
             DrawText ( hdc,
                        "Демонстрация",
                        -1,
                        &rect,
                        DT_SINGLELINE | DT_CENTER | DT_VCENTER );
             EndPaint ( hWnd, &ps );
             return 0;

        case WM_COMMAND:
             /*-------------------------
             Обрабатываем элементы меню.
             -------------------------*/
             switch (wParam)
                    {
                    case IDM_QUIT:
                         /*-----------------------------------
                         Пользователь выбрал команду Завершить.
                         -----------------------------------*/
                         DestroyWindow (hWnd);
                         return 0L;

                    case IDM_ABOUT :
                         /*-------------------------------------
                         Пользователь выбрал команду О программе.
                         -------------------------------------*/
                         DialogBox ( ghInst,
                                     "AboutBox",
                                      hWnd,
                                      lpfnAboutDlgProc );
                         return 0;
                    }/* конец оператора switch(wParam) */

        case WM_DESTROY:
             PostQuitMessage (0);
             return 0;
        }/* конец оператора switch (message) */
```

```
/*----------------------
Сообщение не обработано.
-----------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}/* конец WndProc() */
/*================== конец WndProc() ====================*/

/*======================
 ФУНКЦИЯ: AboutDlgProc()
 ======================*/
/*----------------------------------------
 НАЗНАЧЕНИЕ:
 Это процедура диалогового окна О программе.
 ----------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )

{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;

       case WM_COMMAND :
            switch ( wParam )
                  {
                  case IDOK :
                  case IDCANCEL :
                       EndDialog ( hDlg, 0 );
                       return TRUE;
                  }
       }
return FALSE ;
}/* Конец функции. */
/*=============== Конец AboutDlgProc() ==============*/
```

Листинг 3.3. Generic1.rc

```
/*================================================
 ИМЯ ФАЙЛА: Generic1.rc
 ---------

 НАЗНАЧЕНИЕ ФАЙЛА:
 -----------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 ================================================*/

/*--------
 #include
 --------*/
#include <windows.h>
```

```
#include "Generic1.h"

/*---
 Меню
 ----*/
Generic1  MENU
BEGIN
   POPUP  "&Меню"
       BEGIN
          MENUITEM "&Завершить",  IDM_QUIT
          MENUITEM "&О программе", IDM_ABOUT
       END
END

/*------------------------------
 Определение пиктограммы кассеты.
 Имя файла: Tape.ico
 Имя пиктограммы: IconOfTape
 ------------------------------*/
IconOfTape ICON Tape.ico

/*---------------------------
 Диалоговое окно О программе.
 ---------------------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Об этой программе"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "OK", IDOK, 64, 75, 40, 14
    CTEXT           "(C) Copyright Gurewich 1992, 1993", -1,
                     13, 47, 137, 18
    ICON            "IconOfTape", -1, 14, 12, 18, 20
END
```

Листинг 3.4. Generic1.def

```
;======================================
; Файл определения модуля для Generic1.c
;======================================

NAME          Generic1

DESCRIPTION  'Программа Generic1. (C) Copyright Gurewich 1992, 1993'

EXETYPE       WINDOWS

STUB         'WINSTUB.EXE'

CODE  PRELOAD MOVEABLE DISCARDABLE

DATA  PRELOAD MOVEABLE MULTIPLE

HEAPSIZE      1024
STACKSIZE     8192
```

Листинг 3.5. Generic1.mak

```
#============
#Generic1.mak
#============

Generic1.exe : Generic1.obj Generic1.h Generic1.def Generic1.res
    link /nod Generic1.obj, Generic1.exe, NUL, \
        slibcew.lib oldnames.lib libw.lib commdlg \
        c:\spSDK\TegoWlib\TegoWin.lib, \
        Generic1.def
    rc -t Generic1.res

Generic1.obj : Generic1.c Generic1.h
   cl -c -G2sw -Ow -W3 -Zp Generic1.c

Generic1.res : Generic1.rc Generic1.h Tape.ico
   rc -r Generic1.rc
```

Поскольку все последующие программы, основанные на Generic1, строятся на базе приведенных файлов, рассмотрим их более подробно.

Краткое описание Generic1

Файл Generic1.c располагается в каталоге c:\spSDK\Samp4Win\. Generic1.c — простая программа, тем не менее вы должны ознакомиться с ее листингом и полностью понять ее текст, поскольку, как мы уже упоминали, на ней основаны все последующие программы типа Generic1.

В последующих главах рассматриваются некоторые приложения Windows, в которых реализованы воспроизведение звуковых файлов, синхронизация движения текста и графических объектов с воспроизведением звука (анимация), многозадачность, а также другие интересные темы. При рассмотрении этих тем мы предполагали, что вы знакомы с оболочкой Windows, а также знаете, как писать программы на C под Windows. Так, мы полагаем, что вы уже знаете, что представляют собой WinMain(), WndProc(), файл ресурсов (.RC), файл определения модуля (.DEF), диалоговые окна, командные кнопки и другие понятия, связанные с Windows. Поскольку уровень профессиональной подготовки наших читателей неодинаков, как неодинаков и их опыт в программировании под Windows, мы рассмотрим тексты программы Generic1.c и связанных с ней файлов построчно.

Если вы уже программировали под Windows, то вправе оценить представленный в этой главе материал, как несколько примитивный. В таком случае вам, очевидно, стоит просто пролистать ее, — здесь затронуты несколько новых тем, которые потребуются вам для работы с последующими главами. Таким читателям мы особо рекомендуем прочитать параграф "Библиотека TegoWin.lib."

Если же вы давно не программировали в среде Windows, то материал, изложенный в этой главе, может помочь вам восстановить в памяти необходимые сведения.

Раздел #include программы Generic1.c

Программа Generic1.c содержит три оператора #include:

```
#include <windows.h>
#include "c:\spSDK\TegoWlib\sp4Win.h"
#include "c:\spSDK\Samp4WIN\Generic1.h"
```

Первый #include-файл — windows.h — необходим для всех приложений Windows. Он поставляется вместе с пакетом MS SDK for Windows.

Второй #include-файл — sp4Win.h — располагается в директории c:\spSDK\TegoWLib\. В процессе чтения этой книги вы будете писать программы на C, вызывающие функции C, которые являются частью звуковой библиотеки TegoWin.lib. Прототипы этих звуковых функций C объявлены в #include-файле sp4Win.h. Кроме того, файл sp4Win.h содержит определения и макросы, связанные со звуковой библиотекой TegoWin.lib.

Третий #include-файл — Generic1.h — содержит прототипы функций и определения, характерные для Generic1.c.

Функция WinMain() программы Generic1.c

После операторов #include в программе Generic1.c следует стандартная функция WinMain() с четырьмя параметрами:

```
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
.............................
....  Тело WinMain()  .......
.............................
}
```

Каждое приложение Windows должно содержать функцию WinMain(), так же как каждая программа на C для DOS должна содержать функцию main().

Локальные переменные функции WinMain()

Функция WinMain() содержит три локальные переменные:

```
HWND     hWnd;
MSG      msg;
WNDCLASS  wc;
```

Эти переменные используются в теле функции WinMain() для выполнения различных стандартных инициализаций.

Глобализация экземпляра программы Generic1

Затем программа Generic1.c создает глобальную переменную ghInst, присваивая ей значение hInstance, полученное в качестве первого параметра при вызове функции WinMain() из оболочки Windows:

```
ghInst = hInstance;
```

Глобальная переменная ghInst определена в файле Generic1.h:

```
HINSTANCE  ghInst;
```

Мы объявляем эту переменную глобальной для того, чтобы сделать ее видимой в функции WndProc(), рассматриваемой ниже.

Договоримся, что в качестве первого символа имени глобальной переменной используется буква "g". Например, ghInst — это глобальная переменная.

Обновление и регистрация класса окна

```
if ( !hPrevInstance )
   {
   /*------------------------------------
   Условие if выполнено, это самый первый
   запуск этого приложения.
   ------------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfTape" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW      ) ;
   wc.hbrBackground = GetStockObject ( WHITE_BRUSH );
   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = gszAppName ;

   /*---------------
   Регистрируем окно.
   ------------------*/
   RegisterClass ( &wc );

   }/* конец условия if( !hPrevInstance ) */
```

Обратите внимание на то, что глобальная переменная gszAppName используется для заполнения последних двух элементов структуры wc: lpszMenuName и lpszClassName. Эти строковые переменные были инициализированы в Generic1.h:

```
char gszAppName[] = "Generic1" ;
```

Мы используем имя программы — Generic1 — в качестве имени меню и имени класса (последние два элемента структуры wc).

Создание, отображение и обновление окна программы

Затем мы создаем, отображаем и обновляем окно программы с помощью функций CreateWindow(), ShowWindow() и UpdateWindow():

```
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );

ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );
```

Обратите внимание на то, что мы опять применяем глобальную переменную gszAppName в качестве первых двух параметров функции CreateWindow(). Таким образом, когда вы пишите другие программы, основанные на Generic1, все, что вам необходимо изменить, — это выражение в файле Generic1.h, определяющее переменную gszAppName.

Цикл обработки сообщений программы Generic1.c

Цикл обработки сообщений программы Generic1.c — это стандартный цикл while():

```
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage  ( &msg );
      }
```

Оболочка Windows генерирует сообщения, основываясь на возникающих событиях. Например, если пользователь выбирает элемент меню, то оболочка Windows генерирует сообщение, указывающее на то, что пользователь выбрал данный элемент меню, после чего Windows сохраняет это сообщение в буфере сообщений. А цикл сообщений функции WinMain() извлекает его с помощью функции GetMessage(). Если для нашей программы не существует сообщений, оболочка Windows выполняет функции WinMain() других программ и возвращает управление нашей программе только тогда, когда существует сообщение именно для нашей программы.

Если возникает определенное событие, побуждающее оболочку Windows сгенерировать и сохранить соответствующее ему сообщение для нашей программы (например, пользователь выбрал элемент из меню программы), функция GetMessage() в функции WinMain() извлекает это сообщение; затем выполняется тело цикла while() — сообщение транслируется функцией TranslateMessage(), а после этого возвращается оболочке Windows с помощью функции DispatchMessage(). Получив возвращенное сообщение, Windows выполняет функцию, упомянутую в качестве элемента lpfnWndProc структуры wc. Поскольку мы определили этот элемент как

```
wc.lpfnWndProc   = WndProc;
```

то будет выполнена функция WndProc().

Функция WndProc()

Оболочка Windows вызывает функцию WndProc(), описанную следующим образом:

```
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam)
{
.........................
.... Тело WndProc() ........
.........................
}
```

Прототип функции WndProc() объявлен в файле Generic1.h как

```
long FAR PASCAL _export WndProc ( HWND, UINT, UINT, LONG );
```

Оператор выбора для обработки сообщений в функции WndProc()

Параметры функции WndProc() — это описание сообщения. Функция WndProc() анализирует данное конкретное сообщение и, основываясь на его содержимом, выполняет соответствующую этому сообщению группу операторов. Это достигается использованием оператора switch():

```
switch ( message )
        {
        case WM_CREATE: .........
                        .........
                        .........
                        return 0;

        case WM_PAINT:  .........
                        .........
                        .........
                        return 0;

        case WM_COMMAND:
             switch (wParam)
                    {
                    case IDM_QUIT:   .........
                                     .........
                                     .........
                                     return 0L;

                    case IDM_ABOUT : .........
                                     .........
                                     .........
                                     return 0;
                    }

        case WM_DESTROY: .........
                         .........
                         .........
                         return 0;
        }
return DefWindowProc ( hWnd, message, wParam, lParam ) ;
```

Каждый вариант завершается оператором return 0, указывающим оболочке Windows на то, что данное сообщение обработано. Если сообщение не обработано, то функция WndProc() применяет функцию DefWindowProc().

Обработка сообщения WM_CREATE

При создании окна программы оболочка Windows посылает нашей программе сообщение WM_CREATE. Функция WndProc() обрабатывает это сообщение следующим образом:

```
case WM_CREATE:
     /*---------------------------------------
     Обычно здесь вы открываете звуковой сеанс.
     ---------------------------------------*/
     /*.....................................*/
     /*... Здесь открываем звуковой сеанс ...*/
     /*.....................................*/

     /*-------------------------------------------------------
     Получаем lpfnAboutDlgProc диалогового окна О программе.
     -------------------------------------------------------*/
     lpfnAboutDlgProc =
     MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);
     return 0;
```

В первой части варианта WM_CREATE открывается звуковой сеанс. Фрагмент программы, в котором открывается звуковой сеанс, рассматривается в следующей главе.

Во второй части варианта WM_CREATE создается диалоговое окно команды **О программе**. Мы определяем значение переменной lpfnAboutDlgProc с помощью функции MakeProcInstance(). Переменная lpfnAboutDlgProc позднее используется функцией DialogBox() для создания и отображения диалогового окна команды **О программе** в случае получения сообщения IDM_ABOUT. Обратите внимание на то, что первый параметр вызова функции MakeProcIntance() — это AboutDlgProc. Это означает, что мы назвали функцию диалогового окна AboutDlgProc().

Прототип функции AboutDlgProc() объявлен в Generic1.h как

```
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG );
```

что является стандартным определением для прототипа экспортируемой функции.

Обработка сообщения WM_PAINT

Функция WndProc() обрабатывает сообщение WM_PAINT следующим образом:

```
case WM_PAINT:
     hdc = BeginPaint ( hWnd, &ps );
     GetClientRect ( hWnd, &rect );
     DrawText ( hdc,
                "Демонстрация",
                -1,
                &rect,
                DT_SINGLELINE | DT_CENTER | DT_VCENTER );
     EndPaint ( hWnd, &ps );
     return 0;
```

В варианте WM_PAINT при помощи функции BeginPaint() мы извлекаем контекст устройства hdc, затем получаем клиентную область, используя функцию GetClientRect(), а после этого отображаем в центре окна слово *Демонстрация*.

Оболочка Windows посылает функции WndProc() сообщение WM_PAINT всегда, когда необходимо перерисовать окно нашей программы.

В результате работы операторов, расположенных внутри варианта WM_PAINT, в центре окна **Generic1** отображается слово *Демонстрация*.

Обработка сообщения WM_COMMAND

Функция WndProc() обрабатывает и сообщения WM_COMMAND — сообщения, которые генерируются, если пользователь выбирает одну из команд меню программы.

Когда пользователь выбирает команду из меню, оболочка Windows посылает функции WndProc() сообщение WM_COMMAND. Для сообщения WM_COMMAND мы создаем новый оператор выбора. Этот оператор также проверяет, какая команда выбрана пользователем из меню программы:

```
case WM_COMMAND:
    switch (wParam)
           {
           case IDM_QUIT:
                DestroyWindow (hWnd);
                return 0L;

           case IDM_ABOUT :
                DialogBox ( ghInst,
                            "AboutBox",
                            hWnd,
                            lpfnAboutDlgProc );
                return 0;
           }
```

Как видно из рис. 3.1, меню программы Generic1.c содержит две команды: **Завершить** и **О программе**. В меню приведенных далее программ содержится больше команд, и операторы switch(wParam) с этих программах содержат большее количество вариантов выбора.

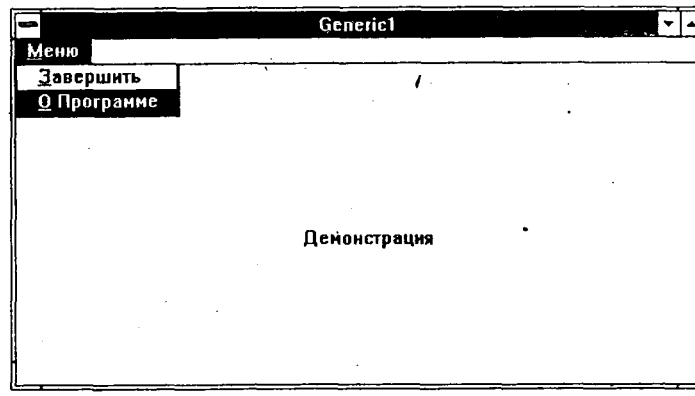


Рис. 3.1. Меню программы Generic1.c

Выбор пользователем команды **Завершить** из меню программы влечет за собой получение функцией WndProc() сообщения WM_COMMAND с параметром wParam, равным IDM_QUIT. Константа IDM_QUIT определена (#define) в файле Generic1.h как 1. В случае IDM_QUIT мы выполняем функцию DestroyWindow(), которая заставляет оболочку Windows в качестве следующего сообщения сгенерировать сообщение WM_DESTROY. Завершаем вариант, возвращая 0, указывающий оболочке Windows на то, что данное сообщение обработано функцией WndProc().

Если пользователь выбирает команду **О программе**, то оболочка Windows выполняет функцию WndProc() с сообщением WM_COMMAND и wParam, равным IDM_ABOUT. Константа IDM_ABOUT определена (#define) в файле Generic1.h как 2. В случае выполнения варианта IDM_ABOUT отображается диалоговое окно **Об этой программе**. Макет этого окна определен в файле ресурсов Generic1.rc. Обратите внимание на то, что четвертый параметр DialogBox() был изменен в варианте WM_CREATE.

Процедура диалогового окна **Об этой программе** — это функция AboutDlgProc():

```
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;

       case WM_COMMAND :
            switch ( wParam )
                   {
                   case IDOK :
                   case IDCANCEL :
                        EndDialog ( hDlg, 0 );
                        return TRUE;
                   }
       }
return FALSE ;
}
```

Прототип AboutDlgProc() объявлен в Generic1.h, а сама функция AboutDlgProc() представляет собой стандартную процедуру обработки диалогового окна с вариантами WM_INITDIALOG и WM_COMMAND.

Сообщение WM_INITDIALOG поступает, когда диалоговое окно создается в первый раз, а сообщение WM_COMMAND — когда пользователь выбирает из меню диалогового окна какую-либо команду.

Обработка сообщения WM_DESTROY

WM_DESTROY — это последнее сообщение, которое обрабатывает функция WndProc():

```
case WM_DESTROY:
     PostQuitMessage (0);
     return 0;
```

Оболочка Windows посылает это сообщение функции WndProc(), когда генерируется запрос на завершение выполнения программы. Такой запрос генерируется в следующих случаях:

1. Пользователь выбрал из системного меню окна нашей программы команду **Закрыть**. Это заставляет оболочку Windows послать функции WndProc() сообщение WM_DESTROY.
2. Пользователь выбрал из меню нашей программы команду **Завершить**. Это заставляет оболочку Windows послать функции WndProc() сообщение WM_COMMAND с wParam, равным IDM_QUIT. Затем обрабатывается вариант IDM_QUIT — вызывается функция DestroyWindow(), заставляющая оболочку Windows послать функции WndProc() сообщение WM_DESTROY.

В случае сообщения WM_DESTROY мы выполняем функцию PostQuitMessage(), которая заставляет генерировать пустое сообщение. Функция GetMessage() принимает пустое сообщение, которое было сгенерировано при выполнении PostQuitMessage(). Тело цикла сообщений в WinMain() не выполняется, и программа завершается.

Файл ресурсов Generic1.rc

Файл ресурсов программы Generic1.c — это файл Generic1.rc.

Он содержит два оператора #include:

```
#include <windows.h>
#include "Generic1.h"
```

Эти #include-файлы необходимы потому, что в файле .RC используются объявленные в них определения (#define).

Следующая часть файла .RC — это определение меню. Меню программы Generic1 содержит два элемента — команды **Завершить** и **О программе** — и определяется следующим образом:

```
Generic1  MENU
    BEGIN
       POPUP  "&Меню"
       BEGIN
            MENUITEM "&Завершить",  IDM_QUIT
            MENUITEM "&О программе", IDM_ABOUT
       END
    END
```

Меню называется Generic1 (как и зафиксировано в WinMain(), где элементу wc.lpszMenuName присвоено значение gszAppName.

Далее в файле .RC следует определение пиктограммы IconOfTape как пиктограммы, хранящейся в файле Tape.ico:

```
IconOfTape ICON Tape.ico,
```

Вы найдете файл Tape.ico в директории c:\spSDK\Samp4Win\. Файл Tape.ico был создан с помощью программы Dialog Image Editor, которая содержится в составе Microsoft SDK for Windows. Эта пиктограмма используется в двух случаях:

1. Как маленький рисунок в диалоговом окне **Об этой программе** (см. рис. 3.2), что указывается в Generic1.rc в строке раздела, определяющего диалоговое окно **Об этой программе**:

```
ICON "IconOfTape", -1, 14, 12, 18, 20
```

2. Как пиктограмма нашей программы, появляющаяся, когда пользователь минимизирует окно нашей программы. Это было задано в функции WinMain(), в которой мы определили элемент wc.hIcon как

```
wc.hIcon  = LoadIcon ( ghInst, "IconOfTape" ) ;
```

Пиктограмма IconOfTape изображена на рис. 3.3.

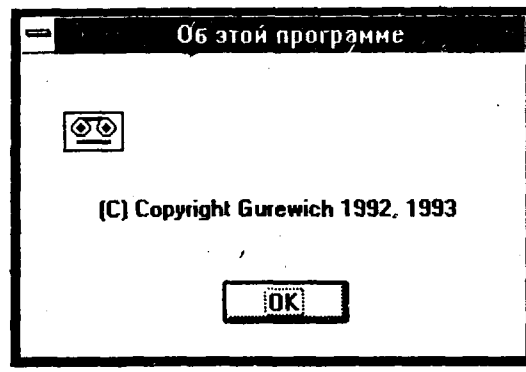


Рис. 3.2. Диалоговое окно Об этой программе

Рис. 3.3. Пиктограмма IconOfTape

В следующем разделе файла .RC определяется структура диалогового окна **Об этой программе**:

```
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Об этой программе"
FONT 8, "MS Sans Serif"
BEGIN
  PUSHBUTTON      "OK", IDOK, 64, 75, 40, 14
  CTEXT           "(C) Copyright Gurewich 1992, 1993", -1,
                   13, 47, 137, 18
  ICON            "IconOfTape", -1, 14, 12, 18, 20
END
```

Файл определения модуля Generic1.def

Для всех приложений Windows требуется файл определения модуля. Файл определения модуля для программы Generic1 — это файл Generic1.def, который расположен в директории c:\spSDK\Samp4Win\.

Файл определения модуля используется в процессе компиляции/компоновки всех приложений Windows.

Make-файл Generic1.mak для компилятора Microsoft

Make-файл программы Generic1.c называется Generic1.mak. Этот файл располагается в директории c:\spSDK\Samp4Win\ и используется для компиляции и компоновки программы Generic1 компилятором Microsoft.

Make-файл Generic1.mak будет выполняться программой NMAKE.EXE, поставляемой в комплекте с пакетом Microsoft C. Программа NMAKE выполняет файл Generic1.mak снизу вверх, а если учесть, что этот файл состоит из трех разделов, то NMAKE вначале будет выполнять третий, затем второй и, наконец, первый раздел.

Последний раздел файла Generic1.rc состоит из следующих строк:

```
Generic1.res : Generic1.rc Generic1.h Tape.ico
     rc -r Generic1.rc
```

Первая строка указывает программе NMAKE.EXE на необходимость выполнить вторую строку раздела в том случае, если файл Generic1.res создан ранее, чем файлы Generic1.rc, Generic1.h или Tape.ico.

Таким образом, модификация любого из указанных файлов — Generic1.rc, Generic1.h или Tape.ico — приведет к выполнению следующей строки:

```
rc -r Generic1.rc
```

Данная строка представляет собой инструкцию по компиляции файла Generic1.rc, то есть указывает на необходимость создания из файла Generic.rc файла Generic1.res.

Ключ -r указывает программе rc на необходимость откомпилировать данный файл, но не компоновать его, то есть не добавлять файл Generic1.res к файлу Generic1.exe (который мы еще не создали).

Второй раздел файла Generic1.mak состоит из следующих строк:

```
Generic1.obj : Generic1.c Generic1.h
     cl -c -G2sw -Ow -W3 -Zp Generic1.c
```

Первая строка данного раздела служит инструкцией программе NMAKE, указывающей на то, что вторая строка раздела должна быть выполнена в том случае, если файл Generic1.obj создан ранее, чем файл Generic1.c или Generic1.h. Таким образом, модификация любого из файлов — Generic1.c или Generic1.h — вызовет выполнение второй строки этого раздела, которая имеет вид

```
cl -c -G2sw -Ow -W3 -Zp Generic1.c
```

где cl — это программа, компилирующая файл Generic1.c для создания объектного файла Generic1.obj.

Назначение ключей, расположенных между словами cl и Generic1.c, приведено ниже.

Ключ -c

Указывает программе cl на необходимость откомпилировать (то есть преобразовать) файл текста программы Generic1.c в объектный файл Generic1.obj, но еще не компоновать его (то есть не создавать исполняемый файл Generic1.exe).

Ключ -G2sw

В программе cl также можно использовать ключи -G2, -Gs и -Gw. Для краткости допускается вводить все три ключа одним словом -G2sw.

Ключ -G2 указывает компилятору cl на необходимость генерировать код, совместимый с ПК на базе процессоров i80286 и более современных — i80386 и i80486.

Ключ -Gs запрещает контроль переполнения стека. В файле Generic1.def мы запросили для стека 8 Кбайт и надеемся на то, что все будет в порядке.

Ключ -Gw указывает программе cl на необходимость добавлять маленькую программу в начало и конец каждой функции, определенной как FAR. Это потребуется при выполнении для перемещения сегментов кода и данных в памяти.

Ключ -Ow

Разрешает компилятору не выполнять некоторые типы оптимизаций. Используя этот ключ, вы будете уверены в том, что сгенерированный объектный код — это именно тот код, который вам нужен (дело в том, что после оптимизации вы зачастую получаете оптимизированный код, несколько отличающийся от того, который вы хотели бы получить).

Ключ -W3

Ключ -W3 устанавливается для "Уровня предупреждения номер 3". При компиляции программа cl выдает вам некоторые предупреждения, которые определены в программе-компиляторе для этого уровня. Не следует игнорировать эти предупреждения: они могут указывать на ошибки, возникающие при выполнении программы.

Ключ -Zp

Данный ключ не предоставляет вам свободы выбора, он просто необходим, так как должен определить способ объявления некоторых структур windows.h. Некоторые из этих структур хранятся выровненными по границе байта, и ключ -Zp указывает компилятору на необходимость упаковывать их, то есть выравнивать по границе байта.

Компонующий раздел файла Generic1.mak

Первый раздел файла Generic1.mak имеет следующий вид

```
Generic1.exe : Generic1.obj Generic1.h Generic1.def Generic1.res
   link /nod Generic1.obj, Generic1.exe, NUL, \
       slibcew.lib oldnames.lib libw.lib commdlg \
       c:\spSDK\TegoWlib\TegoWin.lib, \
       Generic1.def
   rc -t Generic1.res
```

Первая строка — это инструкция, указывающая программе NMAKE на необходимость генерировать исполняемый файл Generic1.exe при условии, что он был создан раньше, чем любой из файлов, следующих за двоеточием (:). Таким образом, модификация любого из файлов — Generic1.obj, Generic1.h, Generic1.def или Generic1.res — приведет к генерации

нового исполняемого файла Generic1.exe. Генерация файла Generic1.exe и определяется этими несколькими строками первого раздела файла Generic1.mak.

Остальные строки первого раздела Generic1.mak имеют следующий вид:

```
link /nod Generic1.obj, Generic1.exe, NUL, \
    slibcew.lib oldnames.lib libw.lib commdlg \
    c:\spSDK\TegoWlib\TegoWin.lib, \
    Generic1.def
rc -t Generic1.res
```

Обратная косая "\" в конце строки указывает на то, что данная команда не окончена и ее продолжение записано в следующей строке.

Первое слово — link — является именем программы, производящей компоновку. Ключ /nod, следующий за словом link, разрешает программе-компоновщику не выполнять поиск библиотек в директориях, заданных по умолчанию, а использовать вместо этого библиотеки, указанные в командной строке link.

За ключом /nod следует имя компонуемого файла — Generic1.obj.

Следующее слово — Generic1.exe — является именем результирующего исполняемого файла.

Затем должно следовать имя файла .MAP, который также может быть создан в результате компоновки. Нам такой файл не нужен, следовательно, мы указываем ключевое слово NUL.

Далее расположен список библиотек, которые должен использовать компоновщик:

1. slibcew.lib
2. oldnames.lib
3. libw.lib
4. commdlg.lib
5. c:\spSDK\TegoWLib\TegoWin.lib

Обратите внимание на то, что список библиотек расположен между двумя запятыми — после слова NUL и после c:\spSDK\TegoWLib\TegoWin.lib.

slibcew.lib Это библиотека Windows для модели памяти small.

oldnames.lib Библиотека, позволяющая использовать нетрадиционные имена функций C, не включенных в библиотеку ANSI C.

libw.lib Это библиотека из пакета Windows SDK. Она необходима для того, чтобы ваша программа могла применять функции из динамически подключаемых библиотек Windows.

commdlg.lib Библиотека стандартных диалоговых окон.

TegoWin.lib Это библиотека, которая располагается в директории c:\spSDK\TegoWLib. Подключив эту библиотеку, ваша программа сможет вызывать находящиеся в ней функции C. Данная библиотека — всего лишь сокращенная версия TegoWin.lib, тем не менее она содержит достаточно функций для компоновки всех примеров программ, приведенных в книге.

Затем следует имя файла Generic1.def, указывающее компоновщику на необходимость связывать файлы в соответствии с файлом Generic1.def.

Последний оператор в разделе компоновки

```
rc -t Generic1.res
```

сообщает компоновщику о необходимости включить в процесс компоновки файл Generic1.res (данный файл, как указывалось в третьем разделе файла Generic1.mak, был ранее сгенерирован из файла Generic1.rc).

Библиотека TegoWin.lib

Для компиляции и компоновки программ для Windows, включенных в эту книгу, вам необходима библиотека "звуковых" функций С. Располагается она в директории c:\spSDK\TegoWLib\ под именем TegoWin.lib и является версией с ограниченными возможностями (сокращенной версией) звуковой библиотеки TegoSoft для компилятора Microsoft C и пакета SDK for Windows. Хотя эта версия и является сокращенной (учтивость фирмы TegoSoft Inc), она, тем не менее, содержит все необходимое программное обеспечение, позволяющее компилировать и компоновать все примеры, приведенные в книге, с помощью компилятора Microsoft C и пакета SDK for Windows.

Еще раз обратите внимание на то, что эта библиотека должна быть использована в процессе компоновки и она действительно упоминается в списке библиотек, которые будут компоноваться (в первом разделе файла .MAK).

Программа NMAKE

Программа NMAKE, поставляемая в комплекте с пакетом Microsoft, работает под управлением файла Generic1.mak. Сначала эта программа выполняет последний раздел инструкций, приведенных в файле Generic1.mak, генерируя файл Generic1.res. Затем она выполняет средний раздел данного файла, генерируя файл Generic1.obj, и, наконец, выполняет первый раздел файла Generic1.mak, генерируя файл Generic1.exe.

Компиляция и компоновка с помощью компилятора Microsoft C

Для компиляции и компоновки программы Generic1 с помощью компилятора Microsoft C выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS введите:

```
NMAKE Generic1.mak    [Enter]
```

Компилятор Microsoft C требует, чтобы ваш ПК в процессе компиляции и компоновки находился в защищенном режиме DOS. Существует два способа перевести ПК в защищенный режим:

1. Запустите Windows, а затем выберите пиктограмму **MS-DOS** из группы программ **Главная**. Так вы получите доступ к DOS. До тех пор, пока вы находитесь в DOS, ваш ПК находится в защищенном режиме и, следовательно, вы можете компилировать и компоновать компилятором Microsoft C.
2. Используйте любую программу-диспетчер памяти, которая позволит перевести ваш ПК в защищенный режим, например Qualitas 386MAX или BlueMAX версии 6.x.

Компиляция и компоновка с помощью программы Programmer Working Bench фирмы Microsoft

Если вы предпочитаете использовать программу Programmer Working Bench (PWB), которая поставляется вместе с компилятором Microsoft, добавьте в список библиотек c:\spSDK\Tego4Win\TegoWin.lib.

Файл компоновки Generic1.mak для компилятора фирмы Borland

Программы, приведенные в книге, можно компилировать и компоновать как компилятором Microsoft C, так и компилятором Borland C. В версиях для Borland используются файлы компоновки с расширением .BMK (например, Generic1.bmk).

Файл компоновки Generic1.mak (для компилятора Microsoft C) похож на файл компоновки Generic1.bmk (для компилятора Borland C). Различия между этими двумя файлами исключительно синтаксические. Например, команда компиляции в файле Generic1.mak — это cl, а в файле Generic1.bmk — bcc.

Для компиляции и компоновки программы Generic1 компилятором Borland выполните следующие действия:

- Войдите в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS введите

```
MAKE -f Generic1.bmk    [Enter]
```

Обратите внимание на то, что компилятор Borland использует программу MAKE.EXE (а не программу NMAKE.EXE), а также на то, что он требует от вас применять ключ -f.

Для компиляции и компоновки всех файлов, независимо от времени и даты их создания, вы можете использовать ключ -B, запустив программу MAKE.EXE следующим образом:

```
MAKE -f Generic1.bmk -B    [Enter]
```

Заметьте, что программа-компилятор Borland C может выполняться в обоих режимах DOS — обычном и защищенном.

Выполнение программы Generic1.exe

Для того чтобы выполнить программу Generic1.exe, можно использовать любой из следующих методов.

Выполнение программы Generic1 из Windows

- Выберите меню **Файл**.
- Выберите из этого меню команду **Выполнить**.
- Введите имя выполняемой программы:

```
c:\spSDK\Samp4Win\Generic1.exe
```

Используйте кнопку **Пролистать** для выбора исполняемого файла Generic1.exe из директории c:\spSDK\Samp4Win\.

Использование пиктограммы для вызова программы

Generic1 — это простая программа, однако другие программы, которые вам предстоит написать, будут более сложными. Вы будете создавать их в несколько этапов: вначале напишете некоторый текст программы, затем откомпилируете и скомпонуете его, после чего попробуете выполнить полученный исполняемый файл. Затем, если вы будете удовлетворены результатами выполнения вашей программы, напишете следующую, более совершенную программу, будете компилировать, компоновать ее — и вновь запустите на выполнение. В таком случае лучше создать пиктограмму для программы, что позволит не тратить время на процесс Файл->Запустить->Пролистать. Вместо этого вы просто выполните двойной щелчок мышью на пиктограмме программы.

Выполнение Generic1 из командной строки DOS

Для выполнения Generic1 из командной строки DOS проверьте, полностью ли вы вышли из Windows (то есть убедитесь в том, что вы не находитесь в окне для выхода в DOS). Затем выполните следующие действия:

- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
WIN Generic1    [Enter]
```

Написание программ, основанных на Generic1.c

Уже было сказано о том, что, используя в качестве шаблонов файлы Generic1, вы можете написать приложения Windows, которые содержат звуковое сопровождение. Вы также можете создать более сложные приложения, основанные на программе Generic2.c.

Убедитесь в том, что вы хорошо понимаете программу Generic1.c и назначение связанных с ней файлов (то есть файлов Generic1.rc, Generic1.h, Generic1.def и Generic1.mak), что вы знаете, как компилировать и компоновать их как с помощью файла Generic1.mac, так и с помощью PWB (Programmer Work Bench). Кроме того, вы должны знать, как выполнить программу Generic1.exe.

Глава 4. Программа Hello.c

Большинство книг, посвященных обучению программированию, начинаются с программы, выводящей фразу "Hello". Мы решили последовать традиции, однако с "незначительным" отличием: наша программа произносит эту фразу.

Сейчас мы напишем наше первое приложение, воспроизводящее звуки, — довольно простую программу для Windows, которая называется Hello.c. Основана она на программе Generic1.

Мы рекомендуем вам, перед тем как приступить к изучению текста программы, откомпилировать и запустить приложение Hello. Это существенно упростит понимание работы программы.

Компиляция и компоновка программы Hello с помощью компилятора фирмы Microsoft

Для компиляции и компоновки программы Hello с помощью компилятора фирмы Microsoft необходимо выполнить следующие действия:

- Убедитесь в том, что ваш компьютер находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS введите:

```
NMAKE Hello.mak    [Enter]
```

Компиляция и компоновка программы Hello с помощью компилятора фирмы Borland

Для компиляции и компоновки программы Hello с помощью компилятора фирмы Borland нужно выполнить следующие действия:

- Войдите в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS введите:

```
MAKE -f Hello.bmk    [Enter]
```

Выполнение программы Hello

Для выполнения программы Hello сделайте следующее:

- Выберите команду **Выполнить** меню **Файл** Диспетчера Программ Windows.
- Воспользуйтесь командной кнопкой **Пролистать** для выбора файла

```
c:\spSDK\Samp4Win\Hello.exe
```

Меню программы Hello содержит три команды: **Играть**, **Завершить** и **О программе**. Главное окно программы Hello.c изображено на рис. 4.1, а диалоговое окно **О программе** — на рис. 4.2.

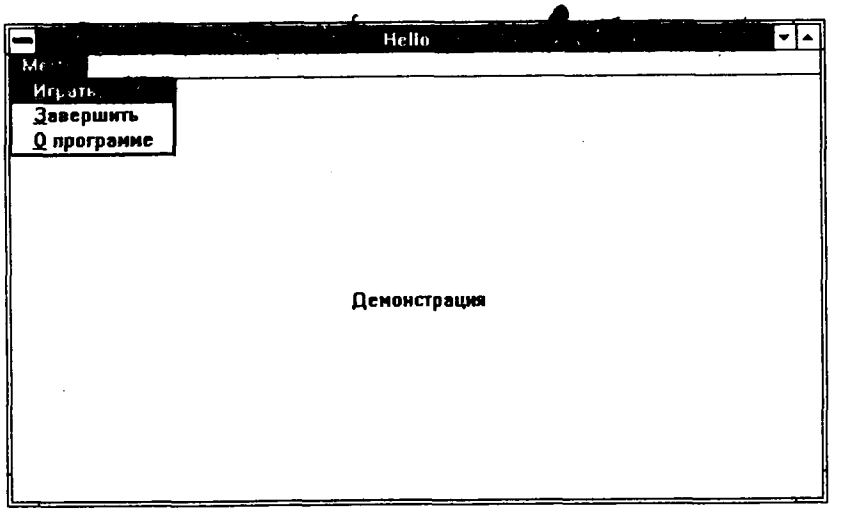


Рис. 4.1. Главное окно программы Hello

Рис. 4.2. Диалоговое окно О программе приложения Hello

Приведем основной алгоритм работы программы Hello:

- Если пользователь выбирает из меню команду **Играть**, то программа воспроизводит фразу "Hello, have a nice day. Good-Bye".
- Если пользователь выбирает из меню команду **О программе**, то программа воспроизводит фразу "Hello" и выводит диалоговое окно **О программе**.
- Если пользователь выбирает из меню команду **Завершить**, то программа воспроизводит фразу "Good-Bye" и заканчивает свою работу.

Файлы, входящие в состав программы Hello

Для программы Hello необходимы следующие файлы:

Hello.c	Текст программы на C
Hello.h	Файл .h
Hello.rc	Файл описания ресурсов программы
Hello.def	Файл определения модуля программы
Hello.mak	Make-файл программы
Tape.ico	Пиктограмма программы

Все эти файлы находятся на прилагаемой дискете и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Листинги файлов программы Hello (листинги 4.1–4.5) приведены ниже.

Листинг 4.1. Hello.h

```
/*========================================================
 ИМЯ ФАЙЛА: Hello.h

 (C) Copyright Gurewich 1992, 1993
 ========================================================*/

/*--------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc      ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG ) ;

/*------
 #define
 -------*/
#define IDM_QUIT  1  /* Команда меню Завершить */
#define IDM_ABOUT 2  /* Команда меню О программе */
#define IDM_PLAY  3  /* Команда меню Играть */

/*--------------------
 Глобальные переменные
 ---------------------*/
char       gszAppName[] = "Hello" ; /* Имя нашего приложения. */
HINSTANCE  ghInst;                  /* текущий экземпляр.     */
```

Листинг 4.2. Hello.c

```
/*============================================================
  ПРОГРАММА: Hello.c
  ----------
  (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

  ОПИСАНИЕ ПРОГРАММЫ:
  -------------------
  Эта программа основана на Generic1.
  В этой программе находится меню Играть.

 ============================================================*/

/*-------
 #include
 --------*/
/*-----------------------------------------
 windows.h  требуется для всех приложений Windows.
 ----------------------------------------+--------*/
#include <windows.h>
```

```
/*-------------------------------------------------------------
 sp4Win.h требуется для того, чтобы функции sp_, макросы SP_
 и #define из звуковой библиотеки могли быть использованы
 в этом приложении.
 -------------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-------------------------------------------------------------
 Определения и прототипы, специфические для данного приложения.
 -------------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\Hello.h"

/*==================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HWND     hWnd; /* Указатель на окно нашего приложения.          */
MSG      msg;  /* Сообщения, обрабатываемые нашим приложением.  */
WNDCLASS wc;   /* Класс окна нашего приложения.                 */

/*-----------------------------------------------
 Создаем экземпляр глобальной переменной hInstance.
 -----------------------------------------------*/
ghInst = hInstance;

/*---------------------------------------
 Инициализируем структуру оконного класса и
 создаем оконный класс.
 ---------------------------------------*/
if ( !hPrevInstance )
   {
   /*-----------------------------------
   Если условие выполняется, то это самый первый
   запуск нашего приложения.
   -----------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfTape" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW      ) ;
   wc.hbrBackground = GetStockObject ( WHITE_BRUSH );
   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = gszAppName ;
```

```
    /*----------------
    Регистрируем окно.
    ------------------*/
    RegisterClass ( &wc );

    }/* конец if ( !hPrevInstance ) */

  /*------------------------------
   Создаем окно нашего приложения.
   ------------------------------*/
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );

/*-------------------------------------------------
 Отображаем и перерисовываем окно нашего приложения.
 -------------------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );

/*--------------
 Цикл сообщений.
 ---------------*/
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage ( &msg );
      }

return msg.wParam ;

} /* Конец функции. */

/*========================= конец WinMain() ====================*/

/*=================
 ФУНКЦИЯ: WndProc()
 ==================*/
/*------------------------------
 ОПИСАНИЕ: обработка сообщений.
 ------------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
```

```
{
/*-----------------------------------
 Локальные и статические переменные.
 -----------------------------------*/
HDC         hdc;    /* Нужен для вывода текстовых сообщений. */
PAINTSTRUCT ps;     /* Нужен для вывода текстовых сообщений. */
RECT        rect;   /* Нужен для вывода текстовых сообщений. */

static FARPROC lpfnAboutDlgProc ; /* Для диалогового окна. */

switch ( message )
       {
       case WM_CREATE:
            /*-------------------------
            Начинаем работу со звуком.
            -------------------------*/
            sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                             SP_NON_STAND_ALONE,
                             0L, /* не применяется. */
                             SP_TS_TYPE);


            /*-------------------------------------------------------
            Получаем lpfnAboutDlgProc диалогового окна О программе.
            -------------------------------------------------------*/
            lpfnAboutDlgProc =
            MakeProcInstance (( FARPROC) AboutDlgProc, ghInst );
            return 0;

       case WM_PAINT:
            hdc = BeginPaint ( hWnd, &ps );
            GetClientRect ( hWnd, &rect );
            DrawText ( hdc,
                       "Демонстрация",
                       -1,
                       &rect,
                       DT_SINGLELINE | DT_CENTER | DT_VCENTER );
            EndPaint ( hWnd, &ps );
            return 0;

       case WM_COMMAND:    /* Обработка команд меню. */
            switch (wParam)
                   {
                   case IDM_PLAY:
                        /*---------------------------------------
                        Проигрываем звуковой файл.
                        Сколько играем: от самого начала до конца.
                        Фраза: "Hello, have a nice day. Good-Bye"
                        ---------------------------------------*/
                        sp_PlayF ( SP_START_OF_FILE, SP_END_OF_FILE );
                        return 0L;

                   case IDM_QUIT:
                        /*---------------------------------------
                        Пользователь выбрал команду меню Завершить.
                        ---------------------------------------*/
```

```
                    DestroyWindow (hWnd);
                    return 0L;

                case IDM_ABOUT :
                    /*-----------------------------------------
                    Пользователь выбрал команду меню О программе.
                    -----------------------------------------*/
                    /*---------------------------------
                    Проигрываем часть звукового файла.
                    Сколько играем: от 0 до 10000
                    Фраза: "Hello"
                    ---------------------------------*/
                    sp_PlayF ( SP_START_OF_FILE, 10000L );

                    DialogBox ( ghInst,
                                "AboutBox",
                                hWnd,
                                lpfnAboutDlgProc );
                    return 0;
                }/* конец switch(wParam) */

        case WM_DESTROY:
            /*-------------------------------
            Проигрываем часть звукового файла.
            Сколько играем: от 30000 до 40000
            Фраза: "Good-Bye"
            -------------------------------*/
            sp_PlayF ( 30000L, 40000L );
            PostQuitMessage (0);
            return 0;
        }/* конец обработки сообщений */

/*----------------------------
 Сообщения не были обработаны.
 ----------------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}
/*================== конец WndProc() ==================*/

/*=======================
 ФУНКЦИЯ: AboutDlgProc()
 =======================*/
/*-----------------------------------------------
 ОПИСАНИЕ:
 Это функция обработки диалогового окна О программе.
 -----------------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
        {
        case WM_INITDIALOG :
            return TRUE;
```

```
        case WM_COMMAND :
             switch ( wParam )
                    {
                    case IDOK :
                    case IDCANCEL :
                         EndDialog ( hDlg, 0 );
                         return TRUE;


return FALSE ;

}/* Конец функции.
/*=============== конец AboutDlgProc() =============*/
```

Листинг 4.3. Hello.rc

```
/*=================================================
 ИМЯ ФАЙЛА: Hello.rc
 ---------

 ОПИСАНИЕ ФАЙЛА:
 ---------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 ================================================*/

/*-------
 #include
 --------*/
#include <windows.h>
#include "Hello.h"

/*---
 Меню
 ----*/
Hello  MENU
BEGIN
   POPUP  "&Меню"
        BEGIN
            MENUITEM "&Играть",  IDM_PLAY
            MENUITEM "&Завершить", IDM_QUIT
            MENUITEM "&О программе", IDM_ABOUT
        END
END

/*-------------------------------------
 Определение пиктограммы с изображением
 магнитофонной кассеты.
 Имя файла: Tape.ico
 Имя пиктограммы: IconOfTape
 -------------------------------------*/
IconOfTape ICON Tape.ico
```

```
/*---------------
 Диалоговое окно.
 ---------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "О программе"
/* необходим локализованный шрифт MS Sans Serif */
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "OK", IDOK, 64, 75, 40, 14
    CTEXT           "(C) Copyright Gurewich 1992, 1993", -1,
                      13, 47, 137, 18
    ICON            "IconOfTape", -1, 14, 12, 18, 20

    CTEXT           "Программа Hello", -1, 46, 13, 79, 12
    ICON            "IconOfTape", -1, 132, 63, 18, 20
    CONTROL         "", -1, "Static", SS_BLACKFRAME, 43, 9, 87, 18

END
```

Листинг 4.4. Hello.def

```
;================================
; Файл определений для Hello.c
;================================

NAME           Hello

DESCRIPTION  'Программа Hello. (C) Copyright Gurewich 1992, 1993'

EXETYPE        WINDOWS

STUB          'WINSTUB.EXE'

CODE   PRELOAD MOVEABLE DISCARDABLE

DATA   PRELOAD MOVEABLE MULTIPLE

HEAPSIZE      1024
STACKSIZE     8192
```

Листинг 4.5. Hello.mak

```
#===========
# Hello.mak
#===========

Hello.exe : Hello.obj Hello.h Hello.def Hello.res
    link /nod Hello.obj, Hello.exe, NUL, \
        slibcew.lib oldnames.lib libw.lib commdlg \
        c:\spSDK\TegoWlib\TegoWin.lib, \
        Hello.def
    rc -t Hello.res
```

```
Hello.obj : Hello.c Hello.h
   cl -c -G2sw -Ow -W3 -Zp Hello.c

Hello.res : Hello.rc Hello.h Tape.ico
   rc -r Hello.rc
```

Make-файл Hello.mak

Файл Hello.mak был создан копированием файла Generic1.mak в Hello.mak и заменой каждого вхождения Generic1 на Hello.

Файл ресурсов Hello.rc

Файл Hello.rc практически идентичен Generic1.rc, с тем отличием, что все вхождения Generic1 заменены на Hello и добавлено меню **Играть**. Изменено также диалоговое окно **О программе**: добавились еще одна пиктограмма — IconOfTape и заключенный в рамку текст *О программе Hello.*

Файл определения модуля Hello.def

Файл Hello.def практически идентичен Generic1.def, с тем отличием, что все вхождения Generic1 заменены на Hello.

#include-файл Hello.h

Файл Hello.h практически идентичен Generic1.h, с тем отличием, что все вхождения Generic1 заменены на Hello и добавлен новый оператор #define:

```
#define IDM_PLAY 3
```

Данный оператор относится к новой команде меню **Играть**.

Файл Hello.c

Файл Hello.c создан копированием файла Generic1.c в Hello.c и заменой каждого вхождения Generic1 на Hello. Кроме того, добавлены несколько функций, которые мы сейчас и рассмотрим.

Функция sp_OpenSession()

В случае получения сообщения WM_CREATE программа Hello.c вызывает функцию sp_OpenSession():

```
sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                 SP_NON_STAND_ALONE,
                 0L, /* не применяется. */
                 SP_TS_TYPE);
```

Таким образом, при создании окна программы мы открываем звуковой сеанс вызовом функции sp_OpenSession(). Прототип функции sp_OpenSession() находится в файле sp4Win.h, который подключается в начале программы.

Функция sp_OpenSession() загружает библиотеку TSEngine для Windows и передает параметры для открытия звукового сеанса. Функция sp_OpenSession() вызывается не только в случае получения сообщения WM_CREATE, но и в любом другом месте программы. Например, некоторые программисты предпочитают вызывать ее из функции WinMain(). Функцию sp_OpenSession() нужно вызывать только один раз (если сеанс предварительно не закрывался), однако, если она будет вызвана несколько раз, то ни к каким критическим последствиям это не приведет. Вызывать функцию sp_OpenSession() один раз рекомендуется потому, что ее выполнение занимает некоторое время и вызов этой функции несколько раз приведет к неоправданным задержкам работы программы. Функция sp_OpenSession() — это функция из библиотеки TSEngine. Как и у всех функций из этой библиотеки, ее название начинается с префикса sp_.

Примечание:

Названия всех функций библиотеки TSEngine начинаются с префикса sp_, например sp_OpenSession().

Ниже мы рассмотрим параметры, передаваемые и возвращаемые функцией sp_OpenSession().

DLL-функции и статические sp_-функции

Обратите внимание на то, что sp_OpenSession() является обыкновенной функцией C для Windows, а не функцией из библиотеки DLL для Windows. Функция sp_OpenSession() включена в библиотеку TegoWin.lib, которая находится в директории c:\spSDK\Tego4Win\.

Как вы знаете из своего опыта создания приложений для Windows, статические библиотеки добавляются к основному модулю на этапе компоновки. Затем они вызываются из основной программы приложения.

Библиотека DLL — это динамическая библиотека, содержащая функции, которые могут вызываться приложениями. В отличие от статических библиотек, библиотеки DLL не включаются в основной код приложения, а находятся в отдельном файле на вашем винчестере. Если при выполнении вашей программы происходит вызов функции из DLL, то соответствующий блок операторов, содержащий эту функцию, динамически загружается в память компьютера и выполняется.

Более подробная информация об использовании динамических звуковых sp_-функций приведена в приложении.

Вы можете спросить — зачем резервируется возможность использования как статических, так и динамических звуковых функций C? Дело в том, что при создании одних приложений лучше использовать статические библиотеки, при создании других — DLL. Предположим, что вы распространяете дискеты с демонстрационным материалом, на которых находятся приложения, использующие звуковое сопровождение. Вы предлагаете пользователю вставить дискету в дисковод, а затем после приглашения DOS ввести **Win demo**. Если это демонстрационное приложение построено с применением статических функций, то его исполняемый код состоит из одного файла и нет необходимости в копировании DLL на винчестер пользователя. Кроме того, вы уверены, что пользователь не потеряет при копировании часть приложения (DLL).

С другой стороны, если ваше приложение достаточно объемно и состоит из нескольких программ, каждая из которых работает со звуком, то надо использовать библиотеку DLL. Этим вы сэкономите значительный объем дискового пространства, поскольку общие функции работы со звуком находятся в отдельном файле — DLL — и используются совместно всеми программами.

Применение остальных sp_-функций

После вызова вашим приложением функции sp_OpenSession() загружается библиотека TSEngine и программа получает возможность воспроизводить музыкальные файлы и выполнять иную работу со звуком с помощью других sp_-функций. Функция sp_OpenSession() должна быть первой из sp_-функций, которые вызывает ваше приложение. И это естественно, так как sp_OpenSession() загружает и активизирует библиотеку TSEngine for Windows.

Примечание:

Функция sp_OpenSession() должна вызываться самой первой из sp_-функций.

Функция sp_OpenSession() может вызываться в любом месте вашей программы.

Рекомендуется размещать вызов функции sp_OpenSession() в блоке обработки события WM_CREATE функции WndProc().

Рекомендуется размещать вызов функции sp_OpenSession() таким образом, чтобы в процессе выполнения программы она вызывалась только один раз.

Параметры функции sp_OpenSession()

В программе Hello.c функции sp_OpenSession() передаются следующие четыре параметра:

```
sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                 SP_NON_STAND_ALONE,
                 0L, /* Не применяется. */
                 SP_TS_TYPE);
```

Звуковой файл Hello.ts

В качестве первого параметра функции передается имя звукового файла — Hello.ts. Этот параметр указывает библиотеке TSEngine на необходимость открыть звуковой сеанс с файлом Hello.ts, который находится в директории c:\spSDK\TSfiles\. При задании пути к звуковому файлу вместо символа "\" нужно использовать "\\", поскольку в языке C символ "\" является управляющим. Если вы уверены в том, что звуковой файл расположен в текущей директории, то в указании пути нет необходимости.

Параметр Stand-Alone

Вторым функции sp_OpenSession() передается целое, именуемое параметром Stand-Alone (автономность). Для того чтобы понять суть этого параметра, предположим, что вы компилируете и компонуете программу Hello.c для ее распространения на дискете. Если вы создадите исполняемый файл программы Hello.exe как неавтономную программу, то на дискете должны быть два файла — Hello.exe и Hello.ts.

Такое решение налагает некоторые ограничения: так, вашему приложению необходимо проверять, установила ли инсталляционная программа файл Hello.ts в директорию, указанную в первом параметре функции sp_OpenSession().

Чтобы избежать этого, можно при компоновке добавить звуковой файл в общий код приложения и таким образом создать автономное приложение. Полученный в результате такой компоновки исполняемый файл Helllo.exe будет содержать и код приложения, и звуковой файл. В этом случае на вашей дискете будет находиться всего один файл —

Hello.exe. Следовательно, второй параметр функции sp_OpenSession() определяет, является ваше приложение автономным или неавтономным.

Задание в качестве второго параметра SP_NON_STAND_ALONE указывает на то, что приложение является неавтономным, а задание SP_STAND_ALONE — что приложение является автономным. Как видите, в программе Hello.c в качестве второго параметра передается SP_NON_STAND_ALONE; следовательно, Hello.exe является неавтономной программой.

Создавать в процессе разработки приложения для Windows неавтономную программу весьма логично, поскольку после того, как разработка будет завершена, достаточно просто заменить данный параметр в вызове функции sp_OpenSession() на SP_STAND_ALONE — и программа будет преобразована в автономную.

Далее вы ознакомитесь с тем, как можно включать звуковой файл в общий код приложения, что даст возможность создавать автономные программы.

Третий параметр функции sp_OpenSession()

Третьим параметром в вызове функции sp_OpenSession() является длинное целое (long). Значение этого параметра всегда должно быть равно 0L.

Параметр File-Type

Четвертым параметром функции sp_OpenSession() передается целое, которое задает тип звукового файла. Для звуковых файлов типа .S четвертый параметр должен быть SP_S_TYPE, а для файлов типа .TS — SP_TS_TYPE. Поскольку в программе Hello.c мы воспроизводим звуковой файл типа .TS, то в качестве четвертого параметра должно передаваться значение SP_TS_TYPE.

Примечание:

Функции sp_OpenSession() нужно передавать следующие четыре параметра:

Первый параметр — строка, оканчивающаяся нулем. Эта строка задает имя звукового файла и путь к нему. Не забудьте, что при указании пути в языке C вместо символа "\" необходимо использовать "\\". Если звуковой файл находится в текущей директории, то путь можно не указывать.

Второй параметр — целое. Он определяет автономность данной программы.
SP_NON_STAND_ALONE = неавтономная программа.
SP_STAND_ALONE = автономная программа.

Третий параметр — длинное целое. Значение этого параметра всегда должно быть равно 0L.

Четвертый параметр — целое. Этот параметр задает тип звукового файла.
SP_S_TYPE = звуковой файл типа .S.
SP_TS_TYPE = звуковой файл типа .TS.
SP_WAV_TYPE = звуковой файл типа .WAV.
SP_VOC_TYPE = звуковой файл типа .VOC.
SP_SND_TYPE = звуковой файл типа .SND.

Функция sp_PlayF()

Для воспроизведения звукового файла или его части программа должна вызывать функцию sp_PlayF().

Буква F в названии этой функции означает Forward (вперед): файл воспроизводится в направлении вперед. Как можно увидеть из прототипа, функции sp_PlayF() передаются два параметра. Прототип функции sp_PlayF() объявлен в файле sp4Win.h, который находится в директории c:\spSDK\TegoWLib\.

Первый передаваемый функции sp_PlayF() параметр — это длинное целое, которое определяет начальную позицию (в байтах) воспроизведения. Последний параметр — тоже длинное целое — определяет конечную позицию воспроизведения. Например, если вы хотите воспроизвести часть звукового файла от позиции 1500 до позиции 7500, то необходимо включить в программу оператор

```
sp_PlayF(1500L, 7500L);
```

Примечание:

Функции sp_PlayF() передаются следующие два параметра:

Первый параметр — длинное целое. Задает начальную позицию (в байтах) воспроизведения части файла.

Второй параметр — длинное целое. Задает конечную позицию (в байтах) воспроизведения части файла.

Функция sp_PlayF() вызывается только после того, как вызвана функция sp_OpenSession().

Проблемой распределения памяти вам заниматься не нужно. Если воспроизводимый участок файла еще не загружен в память, то библиотека TSEngine загрузит его сама, отведя в памяти необходимый участок.

Идентификаторы SP_START_OF_FILE и SP_END_OF_FILE

Когда пользователь выбирает команду меню Играть, оболочка Windows посылает функции WndProc() сообщение WM_COMMAND, а wParam присваивается значение IDM_PLAY. В этом случае IDM_PLAY обрабатывается следующим образом:

```
case IDM_PLAY:
     sp_PlayF (SP_START_OF_FILE, SP_END_OF_FILE);
     return 0L;
```

В качестве параметров функции sp_PlayF() передаются SP_START_OF_FILE и SP_END_OF_FILE. Эти параметры определены в файле sp4Win.h, который находится в директории c:\spSDK\TegoWLib\. Поскольку названия этих констант начинаются с префикса sp_, то легко догадаться, что они принадлежат звуковой библиотеке. Как мы уже упоминали, все макросы и определения из библиотеки звуковой поддержки начинаются с префикса sp_. Например:

```
SP_START_OF_FILE
SP_END_OF_FILE
SP_S_TYPE
SP_NON_STAND_ALONE
```

Определение SP_START_OF_FILE эквивалентно определению 0L и указывает TSEngine на то, что файл нужно воспроизводить с самого начала. Например, приведенные ниже два вызова sp_PlayF() идентичны — оба задают воспроизведение участка звукового файла от начала до 8000:

```
sp_PlayF ( 0L , 8000L );
sp_PlayF ( SP_START_OF_FILE , 8000L );
```

Определение SP_END_OF_FILE указывает TSEngine на то, что файл нужно воспроизводить до самого конца. Так, если длина звукового файла равна 10000000 байт, то приведенные ниже вызовы sp_PlayF() идентичны — оба задают TSEngine воспроизведение файла от позиции 9000 до конца:

```
sp_PlayF ( 9000L , 10000000L );
sp_PlayF ( 9000L , SP_END_OF_FILE );
```

Достоинством использования SP_END_OF_FILE является то, что программе не требуется определять длину воспроизводимого звукового файла. При помощи SP_END_OF_FILE она поручает определить размер этого файла библиотеке TSEngine.

Примечание:

SP_START_OF_FILE — идентификатор SP_START_OF_FILE может передаваться функции sp_PlayF() в качестве первого параметра. Он задает TSEngine воспроизведение звукового файла от начала.

SP_END_OF_FILE — идентификатор SP_END_OF_FILE может передаваться функции sp_PlayF() в качестве второго параметра. Он задает TSEngine воспроизведение звукового файла до конца.

Другие вызовы функции sp_PlayF() в программе Hello.c

Функция sp_PlayF() вызывается также в случае IDM_ABOUT. Когда пользователь выбирает команду меню **О программе**, оболочка Windows посылает функции WndProc() сообщение WM_COMMAND, а wParam устанавливается в IDM_ABOUT. В этом случае сначала воспроизводится участок звукового файла от начала до позиции 10000, а потом выводится диалоговое окно **О программе**:

```
case IDM_ABOUT :
     sp_PlayF ( SP_START_OF_FILE, 10000L );
     DialogBox ( ghInst,
                 "AboutBox",
                 hWnd,
                 lpfnAboutDlgProc );
     return 0;
```

После воспроизведения участка файла от 0 до 10000 выполнение программы продолжается, и вызовом функции DialogBox() выводится диалоговое окно **О программе**.

Вы можете удивиться: откуда мы знаем, что фраза "Hello" находится именно в участке от 0 до 10000? Естественно, эти данные получены не методом проб и ошибок. Дело в том, что для создания звукового файла мы использовали редактор TS Sound Editor, который имеет соответствующий инструментарий для нахождения координат нужных участков.

Программа TS Sound Editor

Редактор TegoSoft (TS) Sound Editor позволяет загружать звуковые файлы и выделять (с помощью мыши или клавиатуры) участки файла. Выбранный участок можно воспроизвести, удалить, перенести, скопировать, вставить в другой звуковой файл и вообще сделать с ним все, что потребуется (на прилагаемой к книге дискете редактора TS Sound Editor нет).

Вообще, TS Sound Editor очень похож на обычный текстовый процессор — ему присущи все основные черты текстового процессора. Хотя, естественно, существуют и некоторые различия. В текстовом процессоре вы обычно выделяете текст, изменяете его начертание или шрифт и т.п. В TS Sound Editor вы выделяете участок звукового файла и вместо того, чтобы утолщать, наклонять и изменять шрифт, производите с этим участком манипуляции, присущие звуку. Вот лишь некоторые из этих манипуляций — увеличение или уменьшение громкости воспроизведения, добавление эха к участку, фильтрование шумовых помех, изменение звука с целью внесения искажений (для создания специальных эффектов), изменение звуковых характеристик для получения эффекта разных голосов.

С помощью TS Sound Editor мы просмотрели файл Hello.ts и нашли, что он состоит из следующих фрагментов:

Начало	*Конец*	*Содержимое*
0	10000	Hello
10000	20000	Have a nice day
30000	40000	Good-Bye

Если у вас есть TS Sound Editor, то вы тоже можете просмотреть любой из своих звуковых файлов.

TS Sound Editor выводит звуковой файл на дисплей в виде синусоиды. Он предоставляет возможность отметить начало и конец любого участка файла, его байтовые координаты, поставить необходимые метки и снабдить воспроизводимую фразу комментарием.

Из приведенного выше описания структуры звукового файла Hello.ts ясно, почему при выборе пользователем директивы меню **О программе** мы воспроизводим участок от 0 до 10000. Этим мы добиваемся воспроизведения фразы "Hello".

Воспроизведение фразы "Good-Bye" при завершении программы

Аналогично, при завершении программы воспроизводится фраза "Good-Bye". Пользователь может завершить приложение выбором команды **Закрыть** из системного меню или команды **Завершить** в основном меню. Если пользователь завершает приложение выбором команды меню **Завершить**, то оболочка Windows посылает функции WndProc() сообщение WM_COMMAND, а wParam устанавливается в IDM_QUIT.

При получении IDM_QUIT вызывается функция DestroyWindow():

```
case IDM_QUIT:
     DestroyWindow(hWnd);
     return 0L;
```

Вызов DestroyWindow(hWnd) влечет за собой передачу управления WndProc(); при этом сообщение равно WM_DESTROY.

При обработке WM_DESTROY мы воспроизводим участок звукового файла, содержащий фразу "Good-Bye":

```
case WM_DESTROY:
     sp_PlayF (30000L, 40000L);
     PostQuitMessage (0);
     return 0;
```

Аналогично, если пользователь завершает приложение из системного меню, то Windows передает управление WndProc(); при этом сообщение равно WM_DESTROY и также воспроизводится фраза "Good-Bye".

Функции sp_PlayLabelF() и sp_PlayTimeF()

Ближайшей "родственницей" функции sp_PlayF() является функция sp_PlayLabelF(). Она значительно более удобна в использовании, чем sp_PlayF(). При воспроизведении участка звукового файла при помощи sp_PlayF() вам нужно точно знать, где начинается и где заканчивается нужный фрагмент, а при использовании функции sp_PlayLabelF() вместо позиций в файле в качестве параметров передаются метки. Эти метки вставляются в звуковой файл при помощи редактора TS Sound Editor. Так, в начале фразы "Good-Bye" вы вставили метку "GoodBye начинается здесь", а в конце — метку "GoodBye заканчивается здесь". И для воспроизведения фразы "Good-Bye" напишете в своей программе следующее:

```
sp_PlayLabelF ("GoodBye начинается здесь", "GoodBye заканчивается здесь");
```

Если вы используете метки, то появляется возможность создать библиотеку с набором наиболее популярных фраз и слов. У вас имеется файл GoodBye.ts, в котором содержится фраза "GoodBye" с соответствующими метками, файл Press.ts, в котором находится фраза "Press any key to continue" с соответствующими метками, а также еще несколько файлов с наиболее популярными фразами.

Мы рекомендуем вам, когда вы будете писать свою программу, создать общий файл для всех фраз. Для объединения всех звуковых файлов используйте TS Sound Editor.

TS Sound Editor позволяет удалить выбранные фрагменты звука, если вы решите, что они вам больше не нужны. Основным достоинством использования меток является то, что можно оперативно воспроизвести нужный фрагмент звукового файла, используя вызов sp_PlayLabelF(), так как этой функции нужно передавать не байтовые позиции, а лишь метки.

При подготовке демонстрационных примеров мы учитывали, что в данный момент у вас, скорее всего, нет редактора TS Sound Editor. Поэтому мы использовали sp_PlayF() чаще, чем sp_PlayLabelF() (тем более, что sp_PlayLabelF() не входит в сокращенную версию звуковой библиотеки TS).

Еще одной близкой "родственницей" функций sp_PlayF() и sp_PlayLabelF() является функция sp_PlayTimeF(), которой также передаются два параметра — вещественные числа. Первый параметр — это начало воспроизводимого фрагмента, задаваемое в секундах, второй — конец фрагмента (тоже в секундах).

Так, для воспроизведения фрагмента звукового файла от временной позиции 7,25 с до 7000,0 с необходимо вызвать функцию sp_PlayTimeF() со следующими параметрами:

```
sp_PlayTimeF (7.25, 7000.0);
```

Поскольку применение этой функции имеет смысл только в том случае, если у вас есть TS Sound Editor, то в демонстрационных примерах для воспроизведения мы в основном использовали sp_PlayF(). Функция sp_PlayTimeF() не содержится в сокращенной версии звуковой библиотеки TS.

На этом мы заканчиваем описание программы Hello.c. Обратите внимание на то, что еще ничего не сказано о переменных, возвращаемых функциями sp_OpenSession() и sp_PlayF(). Этот вопрос освещен в следующей главе.

Рекомендация

Как вы можете помнить из гл. 2, программа NMAKE компилирует и компонует вашу программу в порядке, описанном в файле .MAK. При этом она компилирует и компонует только те файлы, которые были изменены с момента последней компиляции или компоновки. Решение о компиляции NMAKE принимает на основе дат создания подчиненных файлов (файлы, которые перечислены после двоеточия (:) в каждом разделе файла .MAK). При создании приложений Windows файлы зачастую создавались простым копированием (например, в случае Generic1). Иногда в результате копирования и повторного копирования файлов, утилита NMAKE может ошибиться, поскольку при копировании у копии файла сохраняются дата и время создания оригинала.

Если при написании приложения вы в какой-то момент захотите перекомпилировать все файлы вне зависимости от даты и времени их создания, примените ключ /A программы NMAKE, а при использовании компилятора Borland — ключ -B.

Для компилятора Microsoft:

```
NMAKE Hello.mak /A    [Enter]
```

Для компилятора Borland:

```
MAKE -f Hello.bmk -B    [Enter]
```

Вы можете быть уверены, что NMAKE оттранслирует все, что нужно.

Программа Sections

Сейчас мы в качестве упражнения напишем программу Sections.c, которая будет воспроизводить разные фрагменты звукового файла Hello.ts.

Программа Sections имеет следующую спецификацию:

- Программа называется Sections.c.
- Программа основана на Generic1.
- В программе имеется два меню — **Файл** и **Пословно**.
- Характеристики меню **Файл:**

  Название меню: **Файл**

  Команды меню **Файл**:

Команда меню	*Действия*
Играть	Воспроизводится файл Hello.ts.
О программе	Воспроизводится слово "Hello" и выводится диалоговое окно **О программе**.
Завершить	Воспроизводится фраза "Good-Bye" и приложение завершается.

- Характеристики меню **Пословно:**

  Название меню: **Пословно**

  Команды меню **Пословно**:

Команды меню	*Действия*
Hello	Воспроизводится слово "Hello".
Have	Воспроизводится слово "Have".
A	Воспроизводится слово "A".
Nice	Воспроизводится слово "Nice".
Day	Воспроизводится слово "Day".
Good-Bye	Воспроизводится "Good-Bye".

Так, когда пользователь выбирает команду **Nice** в меню **Пословно**, то воспроизводится слово "Nice", а если он выбирает в этом меню команду **Day**, то воспроизводится слово "Day" и т.п.

Меню **Файл** приведено на рис. 4.3, а меню **Пословно** — на рис. 4.4. Диалоговое окно **О программе** показано на рис. 4.5. Оно практически идентично диалоговому окну **О программе** программы Hello.c, с тем отличием, что в него добавлено (просто для красоты) больше значков IconOfTape.

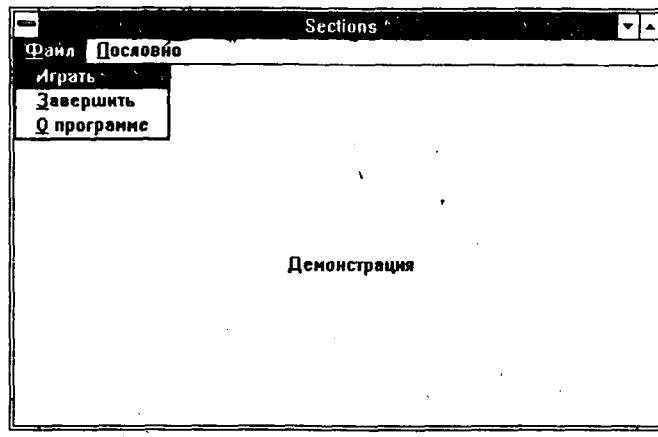


Рис. 4.3. Меню Файл программы Sections.c

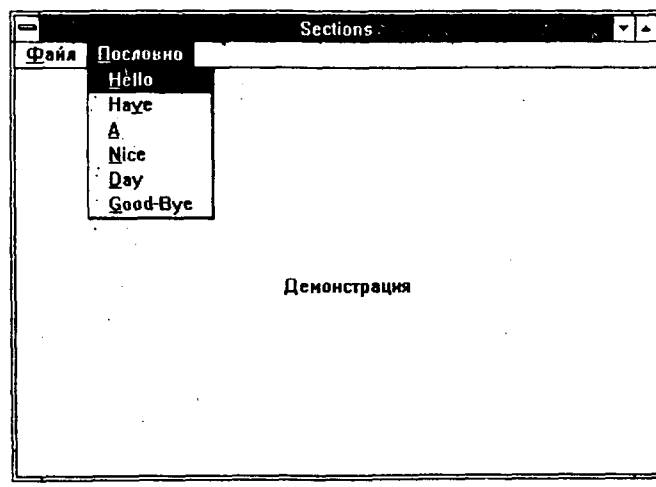


Рис. 4.4. Меню Пословно программы Sections.c

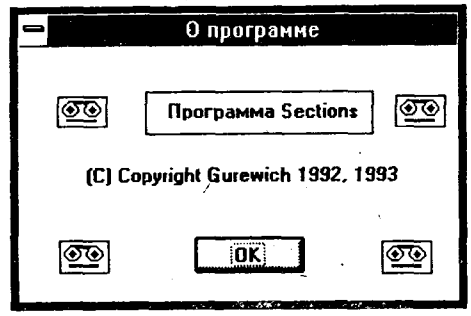


Рис. 4.5. Диалоговое окно О программе приложения Sections

Компиляция и компоновка программы Sections

Для компиляции и компоновки программы Sections с помощью компилятора Microsoft необходимо:

- Убедиться в том, что компьютер находится в защищенном режиме.
- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
NMAKE Sections.mak    [Enter]
```

Для компиляции и компоновки программы Sections с помощью компилятора Borland нужно:

- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
MAKE -f Sections.bmk    [Enter]
```

Выполнение программы Sections

Для выполнения программы Sections необходимо:

- Выбрать команду **Выполнить** из меню **Файл** Диспетчера Программ Windows.
- Использовать диалоговое окно **Пролистать** для выбора файла

```
c:\spSDK\Samp4Win\Sections.exe
```

Звуковые фрагменты в программе Sections

Чтобы написать программу Sections, надо знать позиции каждого фрагмента. Поскольку мы предполагаем, что у вас еще нет редактора TS Sound Editor, то приводим описание всех фрагментов в файле Hello.ts:

Начало	*Конец*	*Содержимое*
0	10000	Hello
15000	18250	Have
18250	18500	A
19000	23000	Nice
30000	40000	Good-Bye

Примечание:

При использовании TS Sound Editor вы затратите на получение этих координат минимальное время.

Файлы, входящие в состав программы Sections

Для программы Sections необходимы следующие файлы:

Sections.c	Текст программы на С
Sections.h	Файл .h
Sections.rc	Описание ресурсов программы
Sections.def	Модуль определений программы
Sections.mak	Make-файл программы
Tape.ico	Пиктограмма программы

Все эти файлы находятся на прилагаемой дискете и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Программа Sections очень похожа на программу Hello. Для более полного понимания ее реализации просмотрите листинги 4.6– 4.10.

Листинг 4.6. Sections.h

```
/*============================================
 ИМЯ ФАЙЛА: Sections.h

 (C) Copyright Gurewich 1992, 1993
 ============================================*/

/*--------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc      ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG ) ;

/*------
 #define
 -------*/
#define IDM_QUIT    1  /* Команда Завершить     в меню Файл */
#define IDM_ABOUT   2  /* Команда О программе   в меню Файл */
#define IDM_PLAY    3  /* Команда Играть        в меню Файл */
#define IDM_HELLO   4  /* Команда Hello         в меню Пословно */
#define IDM_HAVE    5  /* Команда Have          в меню Пословно */
#define IDM_A       6  /* Команда A             в меню Пословно */
#define IDM_NICE    7  /* Команда Nice          в меню Пословно */
#define IDM_DAY     8  /* Команда Day           в меню Пословно */
#define IDM_GOODBYE 9  /* Команда GoodBye       в меню Пословно */

/*--------------------
 Глобальные переменные
 --------------------*/
char       gszAppName[] = "Sections" ; /* Имя нашего приложения. */
HINSTANCE  ghInst;                     /* текущий экземпляр.     */
```

Листинг 4.7. Sections.c

```
/*========================================================================
  ПРОГРАММА: Sections.c
  ---------
  (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

  ОПИСАНИЕ ПРОГРАММЫ:
  -------------------
  Эта программа основана на Generic типа 1.

  В этой программе имеется два меню:
  Меню Файл и меню Пословно.

  В меню Файл имеются следующие команды:
         Играть        .... Играет заданный звуковой файл.

         Завершить     .... Завершает приложение.
         О программе   .... Выводит диалоговое окно О программе.

  В меню Пословно имеются следующие команды:
         Hello      ..... воспроизводится "Hello"
         Have       ..... воспроизводится "have"
         A          ..... воспроизводится "a"
         Nice       ..... воспроизводится "nice"
         Day        ..... воспроизводится "day"
         Good Bye   ..... воспроизводится "Good Bye"

 ========================================================================*/

/*-------
 #include
 --------*/
/*-----------------------------------------------
 windows.h  требуется для всех приложений Windows.
 -----------------------------------------------*/
#include <windows.h>

/*-------------------------------------------------------------
 sp4Win.h требуется для того, чтобы в этом приложении могли
 быть использованы функции sp_, а также макросы SP_ и
 #define из библиотеки звуковой поддержки.
 -------------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-------------------------------------------------------------
 Определения и прототипы, специфические для данного приложения.
 -------------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\Sections.h"

/*==================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
```

```
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HWND     hWnd; /* Указатель на окно нашего приложения.          */
MSG      msg;  /* Сообщения, обрабатываемые нашим приложением. */
WNDCLASS wc;   /* Класс окна для нашего приложения.             */

/*-------------------------------------------------
 Создаем экземпляр глобальной переменной hInstance.
 -------------------------------------------------*/
ghInst = hInstance;

/*---------------------------------------
 Инициализируем структуру оконного класса и
 создаем оконный класс.
 ---------------------------------------*/

if ( !hPrevInstance )
   {
   /*-----------------------------------------
   Если условие выполняется, то это самый первый
   запуск нашего приложения.
   -----------------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfTape" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW      ) ;
   wc.hbrBackground = GetStockObject ( WHITE_BRUSH );
   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = gszAppName ;

   /*----------------
   Регистрируем окно.
   ----------------*/
   RegisterClass ( &wc );

   }/* end of if(!hPrevInstance ) */

/*-------------------------------
  Создаем окно нашего приложения.
 -------------------------------*/
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );
```

```
/*-----------------------------------------------
 Выводим и перерисовываем окно нашего приложения.
 -----------------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );

/*------------------------
 Цикл обработки сообщений.
 ------------------------*/
while ( GetMessage ( &msg, NULL, 0, 0 ) )

      {
      TranslateMessage ( &msg );
      DispatchMessage ( &msg );
      }
return msg.wParam ;
} /* Конец функции. */
/*======================== конец WinMain() ====================*/

/*=================
 ФУНКЦИЯ: WndProc()
 =================*/
/*------------------------------
 ОПИСАНИЕ: обработка сообщений.
 ------------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
{
/*----------------------------------
 Локальные и статические переменные.
 ----------------------------------*/
HDC          hdc;    /* Нужен для вывода текстовых сообщений. */
PAINTSTRUCT  ps;     /* Нужен для вывода текстовых сообщений. */
RECT         rect;   /* Нужен для вывода текстовых сообщений. */

static FARPROC lpfnAboutDlgProc ; /* Для диалогового окна. */

switch ( message )
       {
       case WM_CREATE:
            /*-----------------------
            Начинаем работу со звуком.
            -----------------------*/
            sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                             SP_NON_STAND_ALONE,
                             0L,  /* не используется */
                             SP_TS_TYPE);

            /*-------------------------------------------------
             Получаем lpfnAboutDlgProc диалогового окна О программе.
             -------------------------------------------------*/
             lpfnAboutDlgProc =
             MakeProcInstance (( FARPROC) AboutDlgProc, ghInst );
             return 0;
```

```
    case WM_PAINT:
         hdc = BeginPaint ( hWnd, &ps );
         GetClientRect ( hWnd, &rect );
         DrawText ( hdc,
                    "Демонстрация",
                    -1,
                    &rect,
                    DT_SINGLELINE | DT_CENTER | DT_VCENTER );
         EndPaint ( hWnd, &ps );
         return 0;

    case WM_COMMAND:     /* Обработка команд меню. */
         switch (wParam)
              {
              case IDM_PLAY:
                   /*-----------------------------------------
                   Проигрываем звуковой файл.
                   Сколько играем: от самого начала до конца.
                   Фраза: "Hello, have a nice day. Good-Bye"
                   -----------------------------------------*/
                   sp_PlayF ( SP_START_OF_FILE, SP_END_OF_FILE );
                   return 0L;

              case IDM_QUIT: /* Пользователь выбрал команду */
                             /* меню Завершить.             */
                   DestroyWindow (hWnd);
                   return 0L;

              case IDM_ABOUT :
                   /*--------------------------------
                   Проигрываем часть звукового файла.
                   Сколько играем: от 0 до 10000
                   Фраза: "Hello"
                   --------------------------------*/
                   sp_PlayF ( SP_START_OF_FILE, 10000L );
                   DialogBox ( ghInst,
                               "AboutBox",
                               hWnd,
                               lpfnAboutDlgProc );
                   return 0;

              case IDM_HELLO:
                   /*--------------------------------
                   Проигрываем часть звукового файла.
                   Сколько играем: от 0 до 10000
                   Фраза: "Hello"
                   --------------------------------*/
                   sp_PlayF ( 0L, 10000L );
                   return 0L;

              case IDM_HAVE:
                   /*--------------------------------
                   Проигрываем часть звукового файла.
                   Сколько играем: от 15000 до 18250
                   Фраза: "have"
                   --------------------------------*/
```

```
                    sp_PlayF ( 15000L, 18250L );
                    return 0L;

               case IDM_A:
                    /*---------------------------------
                    Проигрываем часть звукового файла.
                    Сколько играем: от 18250 до 18500
                    Фраза: "a"
                    ---------------------------------*/
                    sp_PlayF ( 18250L, 18500L );
                    return 0L;

               case IDM_NICE:
                    /*---------------------------------
                    Проигрываем часть звукового файла.
                    Сколько играем: от 19000 до 23000
                    Фраза: "nice"
                    ---------------------------------*/
                    sp_PlayF ( 19000L, 23000L );
                    return 0L;

               case IDM_DAY:
                    /*---------------------------------
                    Проигрываем часть звукового файла.
                    Сколько играем: от 23000 до 30000
                    Фраза: "day"
                    ---------------------------------*/
                    sp_PlayF ( 23000L, 30000L );
                    return 0L;

               case IDM_GOODBYE:
                    /*---------------------------------
                    Проигрываем часть звукового файла.
                    Сколько играем: от 30000 до 40000
                    Фраза: "Good-Bye"
                    ---------------------------------*/
                    sp_PlayF ( 30000L, 40000L );
                    return 0L;

               }/* конец switch(wParam) */

          case WM_DESTROY:
               /*---------------------------------
               Проигрываем часть звукового файла.
               Сколько играем: от 30000 до 40000
               Фраза: "Good-Bye".
               ---------------------------------*/
               sp_PlayF ( 30000L, 40000L );
               PostQuitMessage (0);
               return 0;

          }/* конец switch(message) */
```

```
/*----------------------------
 Сообщения не были обработаны.
 ----------------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}
/*================== конец WndProc() ====================*/
/*======================
 ФУНКЦИЯ: AboutDlgProc()
 ======================*/
/*-------------------------------------------------
 ОПИСАНИЕ:
 Это функция обработки диалогового окна О программе.
 -------------------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )

{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;

       case WM_COMMAND :
            switch ( wParam )
                  {
                  case IDOK :
                  case IDCANCEL :
                       EndDialog ( hDlg, 0 );
                       return TRUE;
                  }
       }
return FALSE ;
}/* Конец функции. */
/*=============== конец AboutDlgProc() ==============*/
```

Листинг 4.8. Sections.rc

```
/*================================================
 ИМЯ ФАЙЛА: Sections.rc
 ---------

 ОПИСАНИЕ ФАЙЛА:
 ---------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 ================================================*/

/*-------
 #include
 --------*/
```

```
#include <windows.h>
#include "Sections.h"

/*---
 Меню
 ----*/
Sections  MENU

BEGIN
   POPUP  "&Файл"
       BEGIN
          MENUITEM "&Играть",  IDM_PLAY
          MENUITEM "&Завершить", IDM_QUIT
          MENUITEM "&О программе", IDM_ABOUT
       END
   POPUP  "&Пословно"
       BEGIN
          MENUITEM "&Hello",    IDM_HELLO
          MENUITEM "Ha&ve",     IDM_HAVE
          MENUITEM "&A",        IDM_A
          MENUITEM "&Nice",     IDM_NICE
          MENUITEM "&Day",      IDM_DAY
          MENUITEM "&Good-Bye", IDM_GOODBYE
       END
END

/*-----------------------------------------
 Определение пиктограммы магнитофонной кассеты.
 Имя файла: Tape.ico
 Имя пиктограммы: IconOfTape
 -----------------------------------------*/
IconOfTape ICON Tape.ico

/*---------------
 Диалоговое окно.
 ---------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION  "О программе"
/* необходим русифицированный шрифт MS Sans Serif */
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON       "OK", IDOK, 64, 75, 40, 14
    CTEXT            "(C) Copyright Gurewich 1992, 1993", -1,
                     13, 47, 137, 18
    ICON             "IconOfTape", -1, 14, 12, 18, 20
    CTEXT            "Программа Sections", -1, 48, 21, 83, 10
    CONTROL          "", -1, "Static", SS_BLACKFRAME, 46, 17, 83, 18
    ICON             "IconOfTape", -1, 138, 14, 18, 20
    ICON             "IconOfTape", -1, 132, 71, 18, 20
    ICON             "IconOfTape", -1, 14,  68, 18, 20

END
```

Листинг 4.9. Sections.def

```
;======================================
; Файл определений для Sections.c
;======================================

NAME            Sections

DESCRIPTION     'Программа Sections. (C) Copyright Gurewich 1992, 1993'

EXETYPE         WINDOWS

STUB            'WINSTUB.EXE'

CODE  PRELOAD   MOVEABLE DISCARDABLE

DATA  PRELOAD   MOVEABLE MULTIPLE

HEAPSIZE        1024
STACKSIZE       8192
```

Листинг 4.10. Sections.mak

```
#=============
# Sections.mak
#=============

Sections.exe : Sections.obj Sections.h Sections.def Sections.res
    link /nod Sections.obj, Sections.exe, NUL, \
        slibcew.lib oldnames.lib libw.lib commdlg \
        c:\spSDK\TegoWlib\TegoWin.lib, \
        Sections.def
    rc -t Sections.res

Sections.obj : Sections.c Sections.h
   cl -c -G2sw -Ow -W3 -Zp Sections.c

Sections.res : Sections.rc Sections.h Tape.ico
   rc -r Sections.rc
```

Вариант WM_CREATE

В случае WM_CREATE мы открываем сеанс работы со звуковым файлом Hello.ts и получаем указатель на диалоговое окно **О программе**.

```
   case WM_CREATE:
        /*-------------------------
        Начинаем работу со звуком.
        -------------------------*/
        sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                         SP_NON_STAND_ALONE,
                         0L,  /* Not applicable. */
                         SP_TS_TYPE);

        /*-----------------------------------------------------
         Получаем lpfnAboutDlgProc диалогового окна О программе.
         -----------------------------------------------------*/
```

```
lpfnAboutDlgProc =
MakeProcInstance (( FARPROC) AboutDlgProc, ghInst );
return 0;
```

Сеанс открывается как неавтономная программа и со звуковым файлом TS-типа.

Воспроизведение заданного звукового файла

Если пользователь выбирает команду **Играть** в меню **Файл**, то обрабатывается вариант события IDM_PLAY. Константа IDM_PLAY определена в файле Sections.h. В случае получения IDM_PLAY мы просто воспроизводим заданный файл:

```
case IDM_PLAY:
     /*----------------------------------------
     Проигрываем звуковой файл.
     Сколько играем: от самого начала до конца.
     Фраза: "Hello, have a nice day. Good-Bye"
     ----------------------------------------*/
     sp_PlayF ( SP_START_OF_FILE, SP_END_OF_FILE );
     return 0L;
```

Воспроизведение фрагментов звукового файла

Если пользователь выбрал какую-либо из команд меню **Пословно**, то обрабатывается соответствующее событие. Все команды меню определены в файле Sections.h. Например, если выбрана команда **Have**, то обрабатывается IDM_HAVE:

```
case IDM_HAVE:
     /*--------------------------------
     Проигрываем часть звукового файла.
     Сколько играем: от 15000 до 18250
     Фраза: "have"
     --------------------------------*/
     sp_PlayF ( 15000L, 18250L );
     return 0L;
```

Вариант WM_ABOUT

В случае прихода сообщения WM_ABOUT программа Sections воспроизводит фразу "Hello" (аналогично программе Hello) и выводит диалоговое окно **О программе**:

```
case IDM_ABOUT :
     /*--------------------------------
     Проигрываем часть звукового файла.
     Сколько играем: от 0 до 10000
     Фраза: "Hello"
     --------------------------------*/
     sp_PlayF ( SP_START_OF_FILE, 10000L );
     DialogBox ( ghInst,
                 "AboutBox",
                  hWnd,
                  lpfnAboutDlgProc );
     return 0;
```

Вариант WM_DESTROY

В случае прихода сообщения WM_DESTROY программа Sections воспроизводит фразу "Good-Bye" (аналогично программе Hello) и после этого завершает приложение:

```
case WM_DESTROY:
     /*--------------------------------
     Проигрываем часть звукового файла.
     Сколько играем: от 30000 до 40000
     Фраза: "Good-Bye".
     --------------------------------*/
     sp_PlayF ( 30000L, 40000L );
     PostQuitMessage (0);
     return 0;
```

Программа Push2Say

Программа Push2Say очень похожа на программу Sections, отличие лишь в том, что ее интерфейс более традиционен для приложений Windows.

Главное окно приложения Push2Say изображено на рис. 4.6. Оно состоит из одного меню и трех команд — **Нажать**, **Завершить** и **О программе**.

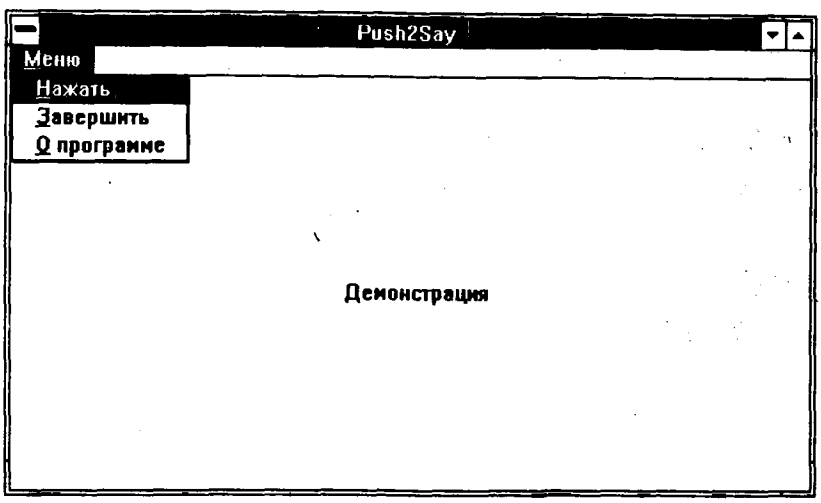


Рис. 4.6. Главное окно приложения Push2Say

Когда пользователь выбирает команду **О программе**, воспроизводится слово "Hello" и выводится диалоговое окно **О программе** (см. рис. 4.8). Когда пользователь выбирает команду **Завершить** — воспроизводится фраза "Good-Bye" и приложение завершается. Когда он выбирает команду **Нажать** — выводится диалоговое окно, содержащее несколько командных кнопок (см. рис. 4.7). Нажатие командной кнопки вызывает воспроизведение соответствующего фрагмента звукового файла. Если нажать клавишу [P] или щелкнуть мышью на кнопке **Нажмите здесь для воспроизведения файла**, то звуковой файл Hello.ts воспроизводится полностью. Если нажать [V] или щелкнуть мышью на кнопке **Have** — воспроизводится слово "Have", при нажатии клавиши [A] воспроизводится звук "A" и т.п.

Компиляция и компоновка программы Push2Say

Для компиляции и компоновки программы Push2Say с помощью компилятора Microsoft необходимо:

- Убедиться, что компьютер находится в защищенном режиме.
- Войти в директорию c:\spSDK\Samp4Win\.

➤ После приглашения DOS ввести:

```
NMAKE Push2Say.mak    [Enter]
```

Для компиляции и компоновки программы Push2Say с помощью компилятора Borland нужно:

➤ Войти в директорию c:\spSDK\Samp4Win\.

➤ После приглашения DOS ввести:

```
MAKE -f Push2Say.bmk    [Enter]
```

Рис. 4.7. Диалоговое окно приложения Push2Say с командными кнопками

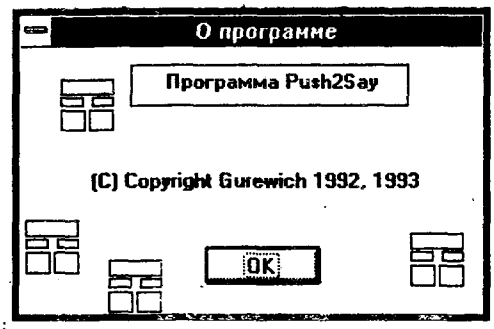


Рис. 4.8. Диалоговое окно О программе приложения Push2Say

Файлы, входящие в состав программы Push2Say

Все файлы данной программы находятся на прилагаемой к книге дискете и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Приведем полный листинг файлов программы Push2Say (см. листинги 4.11–4.15).

Листинг 4.11. Push2Say.h

```
/*============================================
 ИМЯ ФАЙЛА: Push2Say.h

 (C) Copyright Gurewich 1992, 1993
 ============================================*/

/*--------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc      ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export PushDlgProc  ( HWND, UINT, UINT, LONG ) ;

/*------
 #define
 -------*/
#define IDM_QUIT  100  /* Команда меню Завершить. */
#define IDM_ABOUT 101  /* Команда меню О программе. */
#define IDM_PUSH  102  /* Команда меню Нажать. */

#define  PB_HELLO        200  /* Кнопка Hello. */
#define  PB_HAVE         201  /* Кнопка Have.  */
#define  PB_A            202  /* Кнопка A.     */
#define  PB_NICE         203  /* Кнопка Nice.  */
#define  PB_DAY          204  /* Кнопка Day.   */
#define  PB_GOODBYE      205  /* Кнопка Good Bye.    */
#define  PB_ENTIRE_FILE  206  /* Кнопка Entire file. */

/*--------------------
 Глобальные переменные
 ---------------------*/
char       gszAppName[] = "Push2Say" ; /* Имя нашего приложения. */
HINSTANCE  ghInst;                     /* текущий экземпляр.     */
```

Листинг 4.12. Push2Say.c.

```
/*============================================================
  ПРОГРАММА: Push2Say.c
  ----------
  (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

  ОПИСАНИЕ ПРОГРАММЫ:
  -------------------
  Эта программа основана на Generic типа 1.

  Для воспроизведения звука пользователь
  должен войти в меню Нажать.

 ============================================================*/

/*--------
 #include
 ---------*/
```

```
/*-----------------------------------------------
 windows.h  требуется для всех приложений Windows.
 -----------------------------------------------*/
#include <windows.h>

/*-------------------------------------------------------------
 sp4Win.h требуется для того, чтобы функции sp_,  макросы SP_
 и #define из библиотеки звуковой поддержки TS могли быть
 использованы в этом приложении.
 -------------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-------------------------------------------------------------
 Определения и прототипы, специфические для данного приложения.
 -------------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\Push2Say.h"
/*==================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HWND     hWnd; /* Указатель на окно нашего приложения.          */
MSG      msg;  /* Сообщения, обрабатываемые нашим приложением. */
WNDCLASS wc;   /* Класс окна для нашего приложения.             */

/*-----------------------------------------------
 Создаем экземпляр глобальной переменной hInstance.
 -----------------------------------------------*/
ghInst = hInstance;

/*---------------------------------------
 Инициализируем структуру оконного класса и
 создаем оконный класс.
 ---------------------------------------*/
if ( !hPrevInstance )
   {
   /*-----------------------------------------
   Если условие выполняется, то это самый первый
   запуск нашего приложения.
   -----------------------------------------*/
   wc.style           = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc     = WndProc ;
   wc.cbClsExtra      = 0 ;
   wc.cbWndExtra      = 0 ;
   wc.hInstance       = ghInst ;
   wc.hIcon           = LoadIcon   ( ghInst, "IconOfPush2Say" ) ;
   wc.hCursor         = LoadCursor ( NULL, IDC_ARROW      ) ;
   wc.hbrBackground   = GetStockObject ( WHITE_BRUSH );
   wc.lpszMenuName    = gszAppName ;
   wc.lpszClassName   = gszAppName ;
```

```
    /*-----------------
     Регистрируем окно.
     ------------------*/
    RegisterClass ( &wc );

    }/* конец if(!hPrevInstance) */

/*------------------------------
 Создаем окно нашего приложения.
 ------------------------------*/

hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );

/*-----------------------------------------------
 Выводим и перерисовываем окно нашего приложения.
 -----------------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );

/*-------------------------
 Цикл обработки сообщений.
 -------------------------*/
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage ( &msg );
      }

return msg.wParam ;

} /* Конец функции. */
/*========================= конец of WinMain() ====================*/

/*==================
 ФУНКЦИЯ: WndProc()
 ==================*/

/*------------------------------
 ОПИСАНИЕ: обработка сообщений.
 ------------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
```

```
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/

HDC          hdc;    /* Нужен для вывода текстовых сообщений. */
PAINTSTRUCT  ps;     /* Нужен для вывода текстовых сообщений. */
RECT         rect;   /* Нужен для вывода текстовых сообщений. */

static FARPROC lpfnAboutDlgProc ;   /* Для диалогового окна О программе. */
static FARPROC lpfnPushBoxDlgProc ; /* Для диалогового окна Нажать. */

switch ( message )
       {
      case WM_CREATE:
           /*-------------------------
           Начинаем работу со звуком.
           -------------------------*/

           sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                            _SP_NON_STAND_ALONE,
                             0L,   /* не используется     */
                             1);   /* 0 = "S" 1 = "TS"    */

           /*-----------------------------------------------------
           Получаем lpfnAboutDlgProc диалогового окна О программе.
           -----------------------------------------------------*/
           lpfnAboutDlgProc =
           MakeProcInstance (( FARPROC) AboutDlgProc, ghInst );

           /*-----------------------------------------------------
            Получаем lpfnPushBoxDlgProc диалогового окна Нажать.
           -----------------------------------------------------*/
           lpfnPushBoxDlgProc =
           MakeProcInstance (( FARPROC) PushDlgProc, ghInst );
           return 0;

      case WM_PAINT:
           hdc = BeginPaint ( hWnd, &ps );
           GetClientRect ( hWnd, &rect );
           DrawText ( hdc,
                      "Демонстрация",/
                      -1,
                      &rect,
                      DT_SINGLELINE | DT_CENTER | DT_VCENTER );
           EndPaint ( hWnd, &ps );
           return 0;

      case WM_COMMAND:    /* Обработка команд меню. */
           switch (wParam)
                  {
                  case IDM_PUSH:
                       DialogBox ( ghInst,
                                   "PushBox",
                                   hWnd,
                                   lpfnPushBoxDlgProc );
                       return 0L;
```

```
                    case IDM_QUIT: /* Пользователь выбрал      */
                                   /* команду меню Завершить. */
                         DestroyWindow (hWnd);
                         return 0L;

                    case IDM_ABOUT :
                         /*-------------------------------
                         Проигрываем часть звукового файла.
                         Сколько играем: от 0 до 10000
                         Фраза: "Hello"
                         ---------------------------------*/
                         sp_PlayF ( SP_START_OF_FILE, 10000L );
                         DialogBox ( ghInst,
                                     "AboutBox",
                                     hWnd,
                                     lpfnAboutDlgProc );
                         return 0;

                    }/* конец switch(wParam) */

       case WM_DESTROY:
            /*---------------------------------
            Проигрываем часть звукового файла.
            Сколько играем: от 30000 до 40000
            Фраза: "Good-Bye"
            ---------------------------------*/
            sp_PlayF ( 30000L, 40000L );
            PostQuitMessage (0);
            return 0;
       }/* конец switch(message) */

/*----------------------------
 Сообщение не было обработано.
 ----------------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}
/*================== конец of WndProc() ====================*/

/*========================
 ФУНКЦИЯ: AboutDlgProc()
 ========================*/
/*-------------------------------------------------
 ОПИСАНИЕ:
 Это функция обработки диалогового окна О программе.
 -------------------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )

{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;
```

```
        case WM_COMMAND :
             switch ( wParam )
                    {
                    case IDOK :
                    case IDCANCEL :
                         EndDialog ( hDlg, 0 );
                         return TRUE;
                    }
        }
return FALSE ;

}/* Конец функции. */
/*=============== конец AboutDlgProc() ==============*/

/*====================
 ФУНКЦИЯ: PushDlgProc()
 ====================*/

/*-------------------------------------------
 ОПИСАНИЕ:
 Это функция обработки диалогового окна Нажать.
 -------------------------------------------*/

BOOL FAR PASCAL _export PushDlgProc ( HWND hDlg,
                                      UINT message,
                                      UINT wParam,
                                      LONG lParam )
{
switch ( message )
        {
        case WM_INITDIALOG :
             return TRUE;

        case WM_COMMAND :
             switch ( wParam )
                    {
                    case PB_HELLO:
                         /*--------------------------------
                         Проигрываем часть звукового файла.
                         Сколько играем: от 0 до 10000
                         Фраза: "Hello"
                         --------------------------------*/
                         sp_PlayF ( 0L, 10000L );
                         return TRUE;

                    case PB_HAVE:
                         /*--------------------------------
                         Проигрываем часть звукового файла.
                         Сколько играем: от 15000 до 18250
                         Фраза: "have"
                         --------------------------------*/
                         sp_PlayF ( 15000L, 18250L );
                         return TRUE;
```

```
                    case PB_A:
                         /*--------------------------------
                         Проигрываем часть звукового файла.
                         Сколько играем: от 18250 до 18500
                         Фраза: "a"
                         --------------------------------*/
                         sp_PlayF ( 18250L, 18500L );
                         return TRUE;

                    case PB_NICE:
                         /*--------------------------------
                         Проигрываем часть звукового файла.
                         Сколько играем: от 19000 до 23000
                         Фраза: "nice"
                         --------------------------------*/
                         sp_PlayF ( 19000L, 23000L );
                         return TRUE;
                    case PB_DAY:
                         /*--------------------------------
                         Проигрываем часть звукового файла.
                         Сколько играем: от 23000 до 30000
                         Фраза: "day"
                         --------------------------------*/
                         sp_PlayF ( 23000L, 30000L );
                         return TRUE;

                    case PB_GOODBYE:
                         /*--------------------------------
                         Проигрываем часть звукового файла.
                         Сколько играем: от 30000 до 40000
                         Фраза: "Good-Bye"
                         --------------------------------*/
                         sp_PlayF ( 30000L, 40000L );
                         return TRUE;

                    case PB_ENTIRE_FILE:
                         /*----------------------------------------
                         Проигрываем весь звуковой файл.
                         Сколько играем: от самого начала до конца.
                         Фраза: "Hello, have a nice day. Good-Bye"
                         ----------------------------------------*/
                         sp_PlayF ( SP_START_OF_FILE, SP_END_OF_FILE );
                         return TRUE;

                    case IDOK :
                    case IDCANCEL :
                         EndDialog ( hDlg, 0 );
                         return TRUE;
                    }/* конец switch(wParam) */

          }/* конец switch(message) */

return FALSE ;
}/* Конец функции. */
/*================ конец PushDlgProc() ================*/
```

Листинг 4.13. Push2Say.rc

```
/*================================================
 ИМЯ ФАЙЛА: Push2Say.rc
 ---------

 ОПИСАНИЕ ФАЙЛА:
 ---------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 ================================================*/

/*-------
 #include
 --------*/
#include <windows.h>
#include "Push2Say.h"

/*---
 Меню
 ----*/
Push2Say  MENU

BEGIN
   POPUP  "&Меню"
        BEGIN
            MENUITEM "&Нажать",  IDM_PUSH
            MENUITEM "&Завершить", IDM_QUIT
            MENUITEM "&О программе", IDM_ABOUT
        END
END

IconOfPush2Say ICON Push2Say.ico

/*---------------
 Диалоговое окно.
 ----------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION   "О программе"
/* необходим русифицированный шрифт MS Sans Serif */
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON       "OK", IDOK, 64, 75, 40, 14
    CTEXT            "(C) Copyright Gurewich 1992, 1993", -1,
                      13, 47, 137, 18
    ICON             "IconOfPush2Say", -1, 14, 12, 18, 20
    CTEXT            "Программа Push2Say", -1, 45, 9, 84, 14
    CONTROL          "", -1, "Static", SS_BLACKFRAME, 38, 7, 96, 16
    ICON             "IconOfPush2Say", -1, 2, 65, 18, 20
    ICON             "IconOfPush2Say", -1, 31, 80, 18, 20
    ICON             "IconOfPush2Say", -1, 135, 70, 18, 20

END
```

```
/*----------------------
 Диалоговое окно Нажать.
 ----------------------*/
PushBox DIALOG 38, 29, 225, 158
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Нажмите для воспроизведения"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            "IconOfPush2Say", -1,     10,   9, 18, 20
    PUSHBUTTON      "&Hello", PB_HELLO,       45,  52, 40, 14
    PUSHBUTTON      "ha&ve",  PB_HAVE,        96,  52, 40, 14
    PUSHBUTTON      "&a",     PB_A,          148,  52, 40, 14
    PUSHBUTTON      "&nice",  PB_NICE,        70,  73, 40, 14
    PUSHBUTTON      "&day",   PB_DAY,        122,  74, 40, 14
    PUSHBUTTON      "&Good-Bye", PB_GOODBYE, 93,  94, 40, 14
    PUSHBUTTON      "&Нажмите здесь для воспроизведения файла",
                     PB_ENTIRE_FILE, 47, 6, 140, 38
    PUSHBUTTON      "&OK",     IDOK,         60, 124, 40, 27
    PUSHBUTTON      "&Cancel", IDCANCEL,    127, 124, 40, 26
END
```

Листинг 4.14. Push2Say.def

```
;====================================
; Файл определений для   Push2Say.c
;====================================

NAME            Push2Say

DESCRIPTION     'Программа Push2Say. (C) Copyright Gurewich 1992, 1993'

EXETYPE         WINDOWS

STUB            'WINSTUB.EXE'

CODE  PRELOAD   MOVEABLE DISCARDABLE

DATA  PRELOAD   MOVEABLE MULTIPLE

HEAPSIZE      1024
STACKSIZE     8192
```

Листинг 4.15. Push2Say.mak

```
#=============
# Push2Say.mak
#=============

Push2Say.exe : Push2Say.obj Push2Say.h Push2Say.def Push2Say.res
    link /nod Push2Say.obj, Push2Say.exe, NUL, \
        slibcew.lib oldnames.lib libw.lib commdlg \
        c:\spSDK\TegoWlib\TegoWin.lib, \
        Push2Say.def
    rc -t Push2Say.res
```

```
Push2Say.obj : Push2Say.c Push2Say.h
   cl -c -G2sw -Ow -W3 -Zp Push2Say.c

Push2Say.res : Push2Say.rc Push2Say.h Push2Say.ico
   rc -r Push2Say.rc
```

Командные кнопки программы Push2Say

Диалоговое окно программы Push2Say с командными кнопками было создано с помощью программы Dialog Editor, входящей в поставку SDK (Source Development Kit) для Windows (кроме того, в этих целях можно применять и программу Resource Workshop фирмы Borland).

Командные кнопки определены в файле Push2Say.h, а события, возникающие при нажатии этих кнопок, обрабатываются как соответствующие сообщения в процедуре обработки событий диалогового окна **Нажмите для воспроизведения**. Например, сообщение о событии PB_DAY, соответствующее нажатию командной кнопки **Day** (или клавиши [D]), обрабатывается следующим образом:

```
case PB_DAY:
     /*-----------------------------------
     Проигрываем часть звукового файла.
     Сколько играем: от 23000 до 30000
     Фраза: "day"
     ----------------------------------*/
     sp_PlayF ( 23000L, 30000L );
     return TRUE;
```

Обратите внимание на то, что TSEngine воспроизводит звуковые фрагменты без каких-либо задержек.

Глава 5. Типы звуковых файлов

В этой главе описаны различные типы звуковых файлов и продемонстрировано, как можно открыть сеанс работы с ними при помощи вызова функции sp_OpenSession(). Рассматриваются переменные, возвращаемые функцией sp_OpenSession(), которые играют весьма важную роль: они позволяют определить, может ли библиотека TSEngine открыть звуковой сеанс.

Здесь рассмотрены также растровые изображения (bit-map files): как их создавать и отображать. Более подробно эти вопросы освещены в следующих главах, в которых речь идет о программах, использующих анимацию.

Программа FileType

Программа FileType.c воспроизводит звуковые файлы различных типов. На рис. 5.1 изображено главное окно этой программы. Видно, что в нем содержится пять командных кнопок — **S Файл**, **TS Файл**, **WAV Файл**, **VOC Файл** и **Завершить**.

При нажатии пользователем командной кнопки, соответствующей определенному типу звукового файла, воспроизводится файл заданного типа (например, если пользователь нажимает командную кнопку **S Файл**, то воспроизводится звуковой файл с расширением .S). При нажатии командной кнопки **Завершить** приложение завершает свою работу.

Меню программы FileType состоит из двух команд — **Завершить** и **О программе**. Диалоговое окно **О программе** приведено на рис. 5.2.

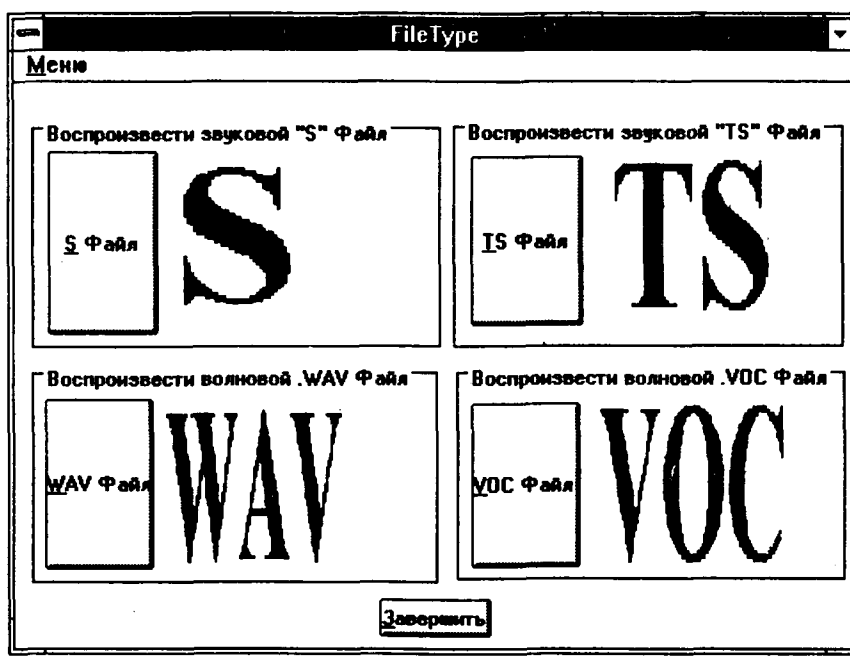


Рис. 5.1. Главное окно приложения FileType

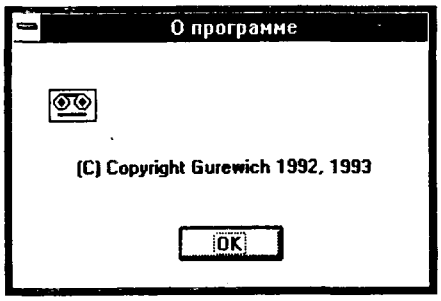


Рис. 5.2. Диалоговое окно О программе

Компиляция и компоновка программы FileType

Для компиляции и компоновки программы FileType с помощью компилятора фирмы Microsoft необходимо выполнить следующие действия:

- Убедиться, что ваш компьютер находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
NMAKE FileType.mak    [Enter]
```

Для компиляции и компоновки программы FileType с помощью компилятора фирмы Borland нужно выполнить следующие действия:

- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
MAKE -f FileType.bmk    [Enter]
```

Выполнение программы Sections

Для выполнения программы Sections необходимо:

- Выбрать в меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- Использовать диалоговое окно **Пролистать** для выбора файла

```
c:\spSDK\Samp4Win\FileType.exe
```

Файлы, входящие в состав программы FileType

Полный комплект программы FileType содержит следующие файлы:

FileType.c	Текст программы на C
FileType.h	Файл .h
FileType.rc	Описание ресурсов программы
FileType.def	Определение модуля программы
FileType.mak	Make-файл программы
Tape.ico	Пиктограмма программы
S.bmp	Растровое изображение символа "S"
TS.bmp	Растровое изображение символа "TS"
WAV.bmp	Растровое изображение символа "WAV"
VOC.bmp	Растровое изображение символа "VOC"
BACKGND.bmp	Растровое изображение, используемое в качестве фона в главном окне приложения

Все эти файлы находятся на прилагаемой дискете и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Все файлы, входящие в программу FileType, приведены в листингах 5.1–5.5.

Листинг 5.1. FileType.h

```
/*=================================================
 ИМЯ ФАЙЛА: FileType.h

 (C) Copyright Gurewich 1992, 1993
 =================================================*/

/*---------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc      ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG );

/*------
 #define
 -------*/
#define IDM_QUIT  1  /* Команда меню Завершить   */
#define IDM_ABOUT 2  /* Команда меню О программе */

#define EXIT_PB 101 /* Кнопка Завершить */
#define S_PB    102 /* Кнопка S         */
#define TS_PB   103 /* Кнопка TS        */
#define WAV_PB  104 /* Кнопка WAV       */
#define VOC_PB  105 /* Кнопка VOC       */

/*--------------------
 Глобальные переменные
 --------------------*/
char       gszAppName[] = "FileType" ; /* Имя нашего приложения. */
HINSTANCE  ghInst;                     /* Текущий экземпляр.     */
```

Листинг 5.2. FileType.c

```
/*=============================================================
  ПРОГРАММА: FileType.c
  ---------
  (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

  ОПИСАНИЕ ПРОГРАММЫ:
  -------------------
  Эта программа основана на Generic типа 1.

  Программа выводит пять кнопок:

     кнопка S ,
     кнопка TS,
     кнопка WAV,
     кнопка VOC,
     кнопка Завершить.
```

```
  Если пользователь нажимает кнопку, проигрывается соответствующий
  звуковой файл; так, если пользователь нажимает кнопку TS,
  воспроизводится звуковой файл типа TS.

  При нажатии пользователем кнопки Завершить приложение
  заканчивает свою работу.

 ============================================================================*/

/*-------
 #include
 --------*/
/*-----------------------------------------------
 windows.h  требуется для всех приложений Windows.
 -----------------------------------------------*/
#include <windows.h>

/*-----------------------------------------------------------------
 sp4Win.h требуется для того, чтобы функции sp_,  макросы SP_ и
 #define из библиотеки звуковой поддержки могли быть использованы
 в этом приложении.
 -----------------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-----------------------------------------------------------
 Определения и прототипы, специфические для данного приложения.
 -----------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\FileType.h"

/*-----------------------------------------------------------
 Для функций C, которым требуются стандартные #include файлы C.
 -----------------------------------------------------------*/
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>

/*=================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*-------------------------------------
 Локальные и статические переменные.
 -------------------------------------*/
HWND     hWnd; /* Идентификатор окна нашего приложения.         */
MSG      msg;  /* Сообщения, обрабатываемые нашим приложением.  */
WNDCLASS wc;   /* Класс окна для нашего приложения.             */

HBITMAP hBitmapOfBkGnd; /* Указатель на растровое изображение фона.*/

HBRUSH  hBrushOfBkGnd;  /* Указатель на кисть фонового изображения.*/
```

```
/*-------------------------------------------------
 Создаем экземпляр глобальной переменной hInstance.
 -------------------------------------------------*/
ghInst = hInstance;

/*-----------------------------------------------------------------
 Загружается фоновое растровое изображение, и создается кисть.
 "BkGndBitmap" определен в файле ресурсов и находится в файле
 BACKGND.bmp.
 -----------------------------------------------------------------*/
hBitmapOfBkGnd = LoadBitmap ( ghInst, "BkGndBitmap" );
hBrushOfBkGnd = CreatePatternBrush ( hBitmapOfBkGnd );

/*---------------------------------------
 Инициализируем структуру оконного класса и
 создаем оконный класс.
 ---------------------------------------*/
if ( !hPrevInstance )
   {
   /*-----------------------------------------
   Если условие выполняется, то это самый первый
   запуск нашего приложения.
   -----------------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = DLGWINDOWEXTRA ;  /* Диалоговое окно в  */
                                        /* качестве главного окна. */
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfTape" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW      ) ;

   wc.hbrBackground = hBrushOfBkGnd;/*Кисть, которую мы создали с */
                                /* помощью растрового изображения.*/

   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = "FileTypeClass" ; /* Как описано в  CLASS */
                                        /* в файле .RC.        */


/*------------------
  Регистрируем окно.
  ------------------*/
  RegisterClass ( &wc );

  }/* конец if( !hPrevInstance ) */

/*-------------------------------------------------------------------
 Создаем безмодальное диалоговое окно.
 Примечание: В этом приложении главным окном является диалоговое окно.
 Его имя FileTypeBox, т.е. как задано в файле описания ресурсов
 (перед ключевым словом DIALOG).
 -------------------------------------------------------------------*/
hWnd = CreateDialog ( ghInst, "FileTypeBox", 0, NULL );
```

```
/*-------------------------------------------------
 Отображаем и перерисовываем окно нашего приложения.
 -------------------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );

/*-------------------------
 Цикл обработки сообщений.
 -------------------------*/
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage ( &msg );
      }

/*---------------------------------------
 Удаляем фоновый объект, который мы создали
 ---------------------------------------*/
DeleteObject ( hBrushOfBkGnd );

return msg.wParam ;

} /* Конец функции. */
/*========================= конец WinMain() ====================*/

/*=================
 ФУНКЦИЯ: WndProc()
 =================*/
/*-----------------------------
 ОПИСАНИЕ: обработка сообщений.
 -----------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
static HDC          hdc;
static HDC          hMemDC;
static PAINTSTRUCT  ps;

static HBITMAP      hBitmapS;   /* Для растрового изображения "S". */
static HBITMAP      hBitmapTS;  /* Для растрового изображения "TS".*/
static HBITMAP      hBitmapWAV; /* Для растрового изображения "WAV".*/
static HBITMAP      hBitmapVOC; /* Для растрового изображения "VOC".*/

static FARPROC lpfnAboutDlgProc ; /* Для диалогового окна О программе. */

static int iOpenResultS;  /* Результат открытия "S" сеанса.  */
static int iOpenResultTS; /* Результат открытия "TS" сеанса. */
```

```
switch ( message )
      {
   case WM_CREATE:
        /*-----------------------------------------------------------
        Загружаем растровые изображения.
        Эти изображения будут позже уничтожены в случае WM_DESTROY.
        -----------------------------------------------------------*/
        hBitmapS   = LoadBitmap ( ghInst, "SBitmap"   );
        hBitmapTS  = LoadBitmap ( ghInst, "TSBitmap"  );
        hBitmapWAV = LoadBitmap ( ghInst, "WAVBitmap" );
        hBitmapVOC = LoadBitmap ( ghInst, "VOCBitmap" );

        /*-----------------------------------------------------
        Получаем lpfnAboutDlgProc диалогового окна О программе.
        -----------------------------------------------------*/
        lpfnAboutDlgProc =
        MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);

        return 0;

   case WM_PAINT:
        hdc = BeginPaint ( hWnd, &ps );
        hMemDC = CreateCompatibleDC ( hdc );

        /*-----------
        Отображаем S.
        -------------*/
        SelectObject ( hMemDC, hBitmapS );
        BitBlt ( hdc,
                 68,
                 31,
                 125,
                 125,
                 hMemDC,
                 0,
                 0,
                 SRCCOPY );

        /*-------------
        Отображаем TS.
        --------------*/
        SelectObject ( hMemDC, hBitmapTS );
        BitBlt ( hdc,
                360,
                31,
                125,
                125,
                hMemDC,
                0,
                0,
                SRCCOPY );

        /*-------------
        Отображаем WAV.
        ---------------*/
        SelectObject ( hMemDC, hBitmapWAV );
```

```
        BitBlt ( hdc,
                 86,
                 180,
                 125,
                 125,
                 hMemDC,
                 0,
                 0,
                 SRCCOPY );

        /*-------------
        Отображаем VOC.
        ---------------*/
        SelectObject ( hMemDC, hBitmapVOC );
        BitBlt ( hdc,
                 360,
                 180,
                 125,
                 125,
                 hMemDC,
                 0,
                 0,
                 SRCCOPY );

        DeleteDC ( hMemDC );
        EndPaint ( hWnd, &ps );

        return 0L;
    case WM_CHAR:
        /*---------------------------
        Получен символ от клавиатуры.
        -----------------------------*/
        /*-----------------------------------
        Преобразуем его к верхнему регистру.
        -------------------------------------*/
        wParam = toupper ( wParam );

        switch ( wParam )
                {
                case 'S':
                     /*-------------------------
                     Пользователь нажимает S для
                     воспроизведения S-файла.
                     -------------------------*/
                     SendMessage ( hWnd,
                                   WM_COMMAND,
                                   S_PB,
                                   0);
                     return 0;

                case 'T':
                     /*-------------------------
                     Пользователь нажимает T для
                     воспроизведения TS-файла.
                     -------------------------*/
```

```
                SendMessage ( hWnd,
                              WM_COMMAND,
                              TS_PB,
                              0);
                return 0;


            case 'W':
                /*-------------------------
                Пользователь нажимает W для
                воспроизведения WAV-файла.
                -------------------------*/
                SendMessage ( hWnd,
                              WM_COMMAND,
                              WAV_PB,
                              0);
                return 0;

            case 'V':
                /*-------------------------
                Пользователь нажимает V для
                воспроизведения VOC-файла.
                -------------------------*/
                SendMessage ( hWnd,
                              WM_COMMAND,
                              VOC_PB,
                              0);
                return 0;

            case 199: /* Код русской буквы З */
                /*-----------------------------------
                Пользователь нажимает З для завершения.
                -----------------------------------*/
                SendMessage ( hWnd,
                              WM_COMMAND,
                              EXIT_PB,
                              0);
                return 0;


            default:
                /*---------------------------------------
                Возвращаемся в Windows, если пользователь
                нажал какую-либо другую клавишу.
                ---------------------------------------*/
                return 0;
                }

case WM_COMMAND:
    /*-----------------------------------------------
    Пользователь выбрал команду меню или нажал кнопку.
    -----------------------------------------------*/
```

```
/*-------------------------------------------
Возвращаем фокус только что выбранного элемента
в hWnd (который является нашим основным окном).
-------------------------------------------*/
SetFocus ( hWnd );

switch (wParam)
        {
        case EXIT_PB:
             /*-----------------------------------
             Пользователь нажал на кнопку Завершить.
             -----------------------------------*/
             DestroyWindow ( hWnd );
             return 0;

        case S_PB:
             /*----------------------------
             Пользователь нажал на кнопку S.
             ----------------------------*/
             iOpenResultS =
             sp_OpenSession ( "c:\\spSDK\\Sfiles\\Sfile.s",
                              SP_NON_STAND_ALONE,
                              0L,
                              SP_S_TYPE );

             /*----------------------------
             Воспроизводим S-файл.
             Сколько играем:
             От: самого начала.
             До: самого конца.
             ----------------------------*/
             if ( iOpenResultS == SP_NO_ERRORS )
                {
                sp_PlayF ( SP_START_OF_FILE,
                           SP_END_OF_FILE );
                }
             else
                {
                MessageBox( NULL,
                           "Не могу открыть S сеанс",
                           "Сообщение от filetype.exe",
                           MB_ICONINFORMATION );
                }
             return 0;

        case TS_PB:
             /*-------------------------------
             Пользователь нажал на кнопку TS.
             -------------------------------*/
             iOpenResultTS =
             sp_OpenSession ( "c:\\spSDK\\TSfiles\\TSfile.ts",
                              SP_NON_STAND_ALONE,
                              0L,
                              SP_TS_TYPE );
```

```
            /*-------------------------
            Воспроизводим TS-файл.
            Сколько играем:
            От: самого начала.
            До: самого конца.
            -------------------------*/

            if ( iOpenResultTS == SP_NO_ERRORS )
               {
               sp_PlayF ( SP_START_OF_FILE,
                          SP_END_OF_FILE );
               }
            else
               {
               MessageBox( NULL,
                          "Не могу открыть TS сеанс",
                          "Сообщение от filetype.exe",
                          MB_ICONINFORMATION );
               }
            return 0;

         case WAV_PB:
            /*-------------------------------
            Пользователь нажал на кнопку WAV.
            -------------------------------*/
            MessageBox( NULL,
                        "Сокращенная версия библиотеки TS
                         не поддерживает WAV файлы.",
                        "Сообщение от filetype.exe",
                        MB_ICONINFORMATION );
            return 0;

         case VOC_PB:
            /*-------------------------------
            Пользователь нажал на кнопку VOC.
            -------------------------------*/
            MessageBox( NULL,
                        "Сокращенная версия библиотеки TS
                        не поддерживает VOC файлы",
                        "Сообщение от filetype.exe",
                        MB_ICONINFORMATION );
            return 0;

         case IDM_QUIT:
            /*-----------------------------------
            Пользователь выбрал команду Завершить
            -----------------------------------*/
            DestroyWindow (hWnd);
            return 0L;

         case IDM_ABOUT :
            /*-------------------------------------
            Пользователь выбрал команду О программе
            -------------------------------------*/
            DialogBox ( ghInst,
                        "AboutBox",
```

```
                                    hWnd,
                                    lpfnAboutDlgProc );
                        return 0;
                    }/* конец switch(wParam) в WM_COMMAND case */

    case WM_DESTROY:
         /*-----------------------------------
         Удаляем объекты растровых изображений.
         -----------------------------------*/
         DeleteObject ( hBitmapS   );
         DeleteObject ( hBitmapTS  );
         DeleteObject ( hBitmapWAV );
         DeleteObject ( hBitmapVOC );

         PostQuitMessage (0);
         return 0;
    }/* конец switch (message) */

/*---------------------------
 Сообщение не было обработано.
 ---------------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;
}
/*================== конец of WndProc() ====================*/

/*======================
 ФУНКЦИЯ: AboutDlgProc()
 ======================*/
/*-------------------------------------------------
 ОПИСАНИЕ:
 Это функция обработки диалогового окна О программе.
 -------------------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
        {
        case WM_INITDIALOG :
             return TRUE;

        case WM_COMMAND :
             switch ( wParam )
                    {
                    case IDOK :

                    case IDCANCEL :
                         EndDialog ( hDlg, 0 );
                         return TRUE;
                    }
        }
return FALSE ;
}/* Конец функции. */
/*=============== конец AboutDlgProc() ==============*/
```

Листинг 5.3. FileType.rc

```
/*=============================================
 ИМЯ ФАЙЛА: FileType.rc
 ----------

 ОПИСАНИЕ ФАЙЛА:
 ---------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 =============================================*/

/*--------
 #include
 --------*/
#include <windows.h>
#include "FileType.h"

/*---
 Меню
 ----*/
FileType  MENU
BEGIN
   POPUP  "&Меню"
        BEGIN
           MENUITEM "&Завершить", IDM_QUIT
           MENUITEM "&О программе", IDM_ABOUT
        END
END

/*-----------------------------------------
 Определение значка магнитофонной кассеты.
 Имя файла: Tape.ico
 Имя значка: IconOfTape
 -----------------------------------------*/
IconOfTape ICON Tape.ico

/*----------------
 Растровые образы.
 -----------------*/
SBitmap     BITMAP S.bmp
TSBitmap    BITMAP TS.bmp
WAVBitmap   BITMAP WAV.bmp
VOCBitmap   BITMAP VOC.bmp
BkGndBitmap BITMAP BACKGND.bmp /* Используется в качестве фона. */

/*---------------------------------------------
 Диалоговое окно О программе данного приложения.
 ---------------------------------------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "О программе"
/* необходим локализованный шрифт MS Sans Serif */
FONT 8, "MS Sans Serif"
```

```
BEGIN
    PUSHBUTTON  "OK", IDOK, 64, 75, 40, 14
    CTEXT       "(C) Copyright Gurewich 1992, 1993", -1,
                13, 47, 137, 18
    ICON        "IconOfTape", -1, 14, 12, 18, 20
END

/*-----------------------------------------------------------------
 Основное диалоговое окно.
 Это диалоговое окно отображается сразу после запуска приложения.
 -----------------------------------------------------------------*/
FileTypeBox DIALOG PRELOAD 28, 25, 299, 220
STYLE WS_MINIMIZEBOX | WS_POPUP   | WS_VISIBLE | WS_CAPTION |
                       WS_SYSMENU | WS_THICKFRAME
CAPTION "FileType"
FONT 8, "MS Sans Serif"
CLASS "FileTypeClass"
BEGIN
    PUSHBUTTON      "&Завершить", EXIT_PB, 129, 190, 40, 14
    PUSHBUTTON      "&S Файл", S_PB, 12, 25, 39, 67
    PUSHBUTTON      "&TS Файл", TS_PB, 162, 27, 40, 62
    GROUPBOX        "Воспроизвести ""TS"" Файл", -1,
                    156, 12, 139, 85
    GROUPBOX        "Воспроизвести ""S"" Файл", -1,
                    6, 13, 145, 84
    GROUPBOX        "Воспроизвести .WAV Файл", WAV_PB,
                    6, 103, 144, 81
    GROUPBOX        "Воспроизвести .VOC Файл", VOC_PB,
                    157, 103, 138, 80
    PUSHBUTTON      "&WAV Файл", WAV_PB,
                    11, 116, 38, 63
    PUSHBUTTON      "&VOC Файл", VOC_PB,
                    162, 118, 39, 60
END
```

Листинг 5.4. FileType.def

```
;====================================
; Файл определенияй для FileType.c
;====================================

NAME            FileType

DESCRIPTION     'Программа FileType. (C) Copyright Gurewich 1992, 1993'

EXETYPE         WINDOWS

STUB            'WINSTUB.EXE'

CODE  PRELOAD MOVEABLE DISCARDABLE

DATA  PRELOAD MOVEABLE MULTIPLE

HEAPSIZE      1024
STACKSIZE     8192
```

Листинг 5.5. FileType.mak

```
#=============
# FileType.mak
#=============

FileType.exe : FileType.obj FileType.h FileType.def FileType.res
    link /nod FileType.obj, FileType.exe, NUL, \
         slibcew.lib oldnames.lib libw.lib commdlg \
         c:\spSDK\TegoWlib\TegoWin.lib, \
         FileType.def
    rc -t FileType.res

FileType.obj : FileType.c FileType.h
  cl -c -G2sw -Ow -W3 -Zp FileType.c

FileType.res : FileType.rc FileType.h Tape.ico
   rc -r FileType.rc
```

Растровые изображения

Содержимое пяти файлов, содержащих растровые изображения (S.bmp, TS.bmp, WAV.bmp, VOC.bmp и BACKGND.bmp), приведено на рис. 5.3.

Рис 5.3. Растровые изображения, используемые в программе FileType: а) файл S.bmp; б) файл TS.bmp; в) файл WAV.bmp; г) файл VOC.bmp; д) Файл BACKGND.bmp .

Краткое описание программы FileType

Целью программы FileType является демонстрация возможностей работы с различными типами звуковых файлов при помощи библиотеки функций С фирмы TS. Кроме того, в ней показано, как программа может использовать переменную, возвращаемую функцией sp_OpenSession().

Применение в качестве главного окна диалогового окна

Обратите внимание на то, что в программе FileType вообще нет главного окна — вместо него применено диалоговое окно, которое появляется сразу после запуска приложения. Это реализуется в функции WinMain() посредством инициализации элемента cbWndExtra структуры wc значением DLGWINDOWEXTRA:

```
wc.cbWndExtra = DLGWINDOWEXTRA
```

и выполнением оператора

```
hWnd = CreateDialog( ghInst, "FileTypeBox", 0, NULL ).
```

Этот оператор создает диалоговое окно в соответствии с классом окна, определенным в элементе lpszClassName структуры wc. В WinMain() мы инициализируем этот элемент как FileTypeClass:

```
wc.lpszClassName = "FileTypeClass".
```

FileTypeClass определен как CLASS в файле FileType.rc:

```
FileTypeBox DIALOG PRELOAD 28, 25, 299, 220
STYLE WS_MINIMIZEBOX | WS_POPUP   | WS_VISIBLE |
      WS_CAPTION | WS_SYSMENU | WS_THICKFRAME
CAPTION "FileType"
FONT 8, "MS Sans Serif"
CLASS "FileTypeClass"
BEGIN
  ..........................
  ..........................
  ..........................
END
```

Обратите внимание на то, что FileTypeBox — это имя диалогового окна и что оно используется в качестве второго параметра при вызове в WinMain() функции CreateDialog().

Главное диалоговое окно содержит меню **FileType** и определяется в WinMain() посредством инициализации элемента lpszMenuName структуры wc значением gszAppName:

```
wc.lpszMenuName = gszAppName;
```

Это меню описывается в файле ресурсов FileType.rc следующим образом:

```
FileType MENU
BEGIN
  ..........................
  ..........................
  ..........................
END
```

Обработка сообщений WM_COMMAND

Командные кнопки в файле описаний FileType.h определены следующим образом: S_PB (командная кнопка **S**), TS_PB (командная кнопка **TS**), WAV_PB (командная кнопка **WAV**), VOC_PB (командная кнопка **VOC**) и EXIT_PB (командная кнопка **Завершить**), а их позиции заданы в файле ресурсов FileType.rc.

Если в качестве главного окна программы используется диалоговое окно, то процедурой, обслуживающей это диалоговое окно является функция WndProc(). Мы поручаем обработку сообщений функции WndProc() в WinMain(), когда инициализируем элемент lpfnWndProc структуры wc следующим значением:

```
wc.lpfnWndProc = WndProc;
```

Все события, имевшие отношение к диалоговому окну (например, нажатие командной кнопки), обрабатываются в функции WndProc() внутри варианта WM_COMMAND:

```
case WM_COMMAND
     {
     case S_PB:
          /*---------------------------------------
          Производим действия, которые необходимы при
          нажатии кнопки S
          ---------------------------------------*/
```

```
                .....................
                .....................
                .....................
                Обработка других кнопок и команд меню.

        }
```

Группы операторов, которые отвечают за обработку командных кнопок **S** и **TS**, открывают звуковой сеанс и воспроизводят звуковой файл соответствующего типа. Естественно, что в случае **S** в вызове функции sp_OpenSession() четвертым параметром является SP_S_TYPE, а случае **TS** — SP_TS_TYPE.

Вот как выглядит вызов sp_OpenSession() в случае TS-сеанса:

```
iOpenResultTS =
sp_OpenSession("c:\\spSDK\\TSfiles\TSfile.ts",
                SP_NON_STAND_ALONE,
                0L,
                SP_TS_TYPE);
```

Значение, возвращаемое функцией sp_OpenSession(), мы присваиваем целым переменным iOpenResultS и iOpenResultTS соответственно. После этого проверяем содержимое iOpenResultS или iOpenResultTS. В приведенном ниже фрагменте программы проверяется значение, возвращаемое при открытии TS-сеанса:

```
if ( iOpenResultTS == SP_NO_ERRORS )
   {
   sp_PlayF ( SP_START_OF_FILE,
              SP_END_OF_FILE);
   }
else
   {
   MessageBox  ( NULL,
                 "Не могу открыть S сеанс.",
                 "Сообщение от filetype.exe",
                 MB_ICONINFORMATION );
}
```

Если функция sp_OpenSession() возвратила значение SP_NO_ERRORS, то это означает, что звуковой сеанс открыт успешно и, следовательно, мы можем воспроизвести S-файл при помощи функции sp_PlayF().

Если значение, возвращенное функцией sp_OpenSession(), отличается от SP_NO_ERRORS, это означает, что во время открытия звукового сеанса произошла какая-то ошибка. В таком случае мы ничего не воспроизводим, а выводим окно сообщения.

Очень важно понять, что если звуковой сеанс не был успешно открыт, то ваша программа не должна вызывать функцию sp_PlayF(). А если вы все таки попытаетесь что-либо воспроизвести при неоткрытом сеансе, то TSEngine просто проигнорирует ваш запрос.

Полная версия библиотеки TS C позволяет открывать звуковой сеанс с WAV-файлами, VOC-файлами и SND-файлами, сокращенная версия поддерживает работу только с S- и TS-файлами. Это ограничение приводит к выводу окна сообщения при нажатии пользователем командной кнопки **WAV** или **VOC**. Если же в вашем распоряжении полная версия библиотеки функций C фирмы TS, то вам достаточно просто убрать вывод окна сообщения и вставить соответствующие операторы для воспроизведения файлов типа WAV и VOC. Приведем фрагмент программы для случая WAV_PB:

```
iOpenResultWAV =
sp_OpenSession ( "c:\\spSDK\\WAVfiles\\WAVfile.wav",
                  SP_NON_STAND_ALONE,
                  0L,
                  SP_WAV_TYPE );
```

```
if ( iOpenResultWAV == SP_NO_ERRORS )
   {
   sp_PlayF ( SP_START_OF_FILE,
              SP_END_OF_FILE );
   }
else
   {
   MessageBox( NULL,
               "Не могу открыть WAV сеанс",
               "Сообщение от filetype.exe",
                MB_ICONINFORMATION );
   }
```

Варианту VOC_PB соответствует следующий фрагмент:

```
iOpenResultVOC =
sp_OpenSession ( "c:\\spSDK\\VOCfiles\\VOCfile.voc",
                  SP_NON_STAND_ALONE,
                  0L,
                  SP_VOC_TYPE );
if ( iOpenResultVOC == SP_NO_ERRORS )
   {
   sp_PlayF ( SP_START_OF_FILE,
              SP_END_OF_FILE );
   }
else
   {
   MessageBox( NULL,
               "Не могу открыть VOC сеанс.",
               "Сообщение от filetype.exe",
               MB_ICONINFORMATION );
   }
```

Полная версия библиотеки функций С фирмы TS поддерживает также работу с SND-файлами. Для таких файлов в качестве четвертого параметра функции sp_OpenSession() передается SP_SND_TYPE.

Другие значения, возвращаемые функцией sp_OpenSession()

Как уже было отмечено, возврат функцией sp_OpenSession() значения SP_NO_ERRORS означает, что сеанс открыт успешно.

Если же сеанс не был открыт, то программа имеет возможность определить причину неудачи. Это может очень пригодиться в процессе разработки программы. Ниже приведены возможные значения, возвращаемые функцией sp_OpenSession().

Значение, возвращаемое функцией sp_OpenSession()	*Расшифровка*
SP_NO_ERRORS	Ошибок нет. Звуковой сеанс открыт успешно.
SP_CANT_OPEN_FILE	Ошибка. Невозможно открыть файл. Например, звуковой файл должен был находиться на дискете в дисководе A:, но в данном дисководе отсутствует дискета.
SP_FILESIZE_ERROR	Ошибка. TSEngine не может определить размер файла. Причиной возникновения этой ошибки может стать сбойный сектор на жестком диске.

SP_ALLOC_FAILURE	Ошибка. Нехватка памяти. Эта ошибка встречается очень редко, особенно под MS Windows.
SP_READ_ERROR	Ошибка. TSEngine не может прочитать файл. Возможно, в теле файла обнаружен сбойный сектор или содержимое звукового файла запорчено.
SP_UNSUPPORTED_TYPE	Ошибка. TSEngine не может определить тип звукового файла.

Примечание:

Если TSEngine не может открыть звуковой сеанс, то наиболее распространенной причиной является либо отсутствие звукового файла в указанном месте, либо некорректность звукового файла.

Обработка сообщений от клавиатуры

Клавиши активизации командных кнопок приложения (так называемые горячие клавиши) обрабатываются в варианте WM_CHAR в функции WndProc().

```
case WM_CHAR:
     wParam = toupper ( wParam );
     {
     case 'S':
          /*-----------------------------------------
           Обрабатываем соответствующим образом нажатие
           клавиши [S] или [s]
          -----------------------------------------*/
     ..................................................
     ..................................................
     ..................................................
     ..................................................
     .....остальные варианты обработки нажатия клавиш.....
     ..................................................

     default: return 0;
     }
```

Обратите внимание на то, что сначала нужно преобразовать полученный символ в соответствующий символ верхнего регистра посредством вызова функции toupper().

По умолчанию (default) в варианте WM_CHAR возвращается 0, поскольку если пользователь нажал не [E], [S], [T], [W] или [V], то, возможно, это нажатие было командой непосредственно оболочке Windows и Windows необходимо ее соответствующим образом обработать.

Для каждого варианта обработки WM_CHAR мы выполняем вызов функции SendMessage(), посылая обработчику событий сообщение о нажатии соответствующей командной кнопки приложения. Например, если пользователь нажал клавишу [S], желая активизировать таким образом командную кнопку S приложения, то посылается следующее сообщение:

```
SendMessage ( hWnd,
              WM_COMMAND,
              S_PB,
              0 );
```

Диалоговое окно О программе

Если из меню выбирается команда **О программе**, то выводится диалоговое окно **О программе**. Внешний вид этого окна описан в файле FileType.rc.

Выбор из меню команды **О программе** обрабатывается в варианте сообщения WM_ABOUT в обработке событий WM_COMMAND.

Функция SetFocus()

Обратите внимание на то, что в самом начале обработки WM_COMMAND мы вызываем функцию SetFocus():

```
SetFocus (hWnd);
```

Этот оператор применяется потому, что после нажатия пользователем командной кнопки на эту кнопку устанавливается фокус, а значит, другие нажатые командные кнопки, например **S** или **T**, не будут обрабатываться WndProc(), поскольку фокус установлен на нажатую ранее командную кнопку и следующую нажатую клавишу должна будет обрабатывать собственная WndProc() данной командной кнопки. Вызовом же функции SetFocus (hWnd) мы устанавливаем фокус на главное диалоговое окно. И при последующем нажатии клавиши Windows будет посылать сообщение WM_CHAR функции WndProc(); при этом wParam присваивается значение, соответствующее нажатой клавише.

Отображение растровых изображений

В следующей главе речь идет о написании программы, использующей анимацию (движение картинки на фоновом изображении одновременно с воспроизведением звука). Поэтому следует знать, как создавать и воспроизводить растровые изображения. Кроме того, вам потребуются базовые знания о растровых изображениях. Эти вопросы освещены в последующих главах.

Рассмотрим создание и воспроизведение растровых изображений на примере программы FileType.

Программа FileType выводит четыре растровых изображения: S.bmp, TS.bmp, WAV.bmp и VOC.bmp. Они выводятся в варианте WM_PAINT. Вызовом функции BeginPaint() получаем hdc , а вызовом функции CreateCompatibleDC() — hMemDC. После того как мы получили hdc и hMemDC, можно вызывать функцию BitBlt() для вывода растровых изображений:

```
case WM_PAINT:
    hdc = BeginPaint ( hWnd, &ps );
    hMemDC = CreateCompatibleDC ( hdc );

    /*--------------
    Отображаем "S".
    ---------------*/
    SelectObject ( hMemDC, hBitmapS );
    BitBlt ( hdc,
             68,
             31,
             125,
             125,
             hMemDC,
             0,
             0,
             SRCCOPY );
```

```
.....
.... Вывод остальных растровых изображений ....
.....

DeleteDC ( hMemDC );
EndPaint (hWnd, &ps );

return 0L;
```

Четвертый и пятый параметры, передаваемые функции BitBlt(), — это ширина и высота растрового рисунка соответственно. В качестве значений этих параметров мы указали 125.

Вот как мы создали S.bmp:

- Использовав программу Paintbrush, нарисовали изображение "S" (см. рис. 5.3).
- Затем сохранили это изображение в виде растрового рисунка (S.bmp).

Аналогично были созданы другие растровые рисунки.

Как мы получили координаты *x* и *y*? При создании диалогового окна FileTypeBox мы использовали Dialog Editor, который входит в поставку SDK для Windows. В диалоговом окне расположили необходимые командные кнопки и туда же поместили четыре указанных пиктограммы — в те места, где впоследствии предполагали разместить растровые рисунки. Затем просмотрели соответствующий DLG-файл, созданный Dialog Editor, и запомнили координаты *x* и *y* указанных пиктограмм. При создании файла ресурсов FileType.rc из DLG-файла мы удалили эти четыре пиктограммы (они больше не нужны) и применили их координаты в качестве второго и третьего параметров в вызове функции BitBlt().

Функция BitBlt() не использует имени BMP-файла. Для S.bmp она, например, использует hBitmapS. Инициализируем hBitmapS (и другие растровые рисунки) в варианте WM_CREATE функции WndProc() следующим образом:

```
hBitmapS   = LoadBitmap ( ghInst, "SBitmap"   );
hBitmapTS  = LoadBitmap ( ghInst, "TSBitmap"  );
hBitmapWAV = LoadBitmap ( ghInst, "WAVBitmap" );
hBitmapVOC = LoadBitmap ( ghInst, "VOCBitmap" );
```

Имена указанных растровых файлов определены в файле FileType.rc так:

```
SBitmap     BITMAP S.bmp
TSBitmap    BITMAP TS.bmp
WAVBitmap   BITMAP WAV.bmp
VOCBitmap   BITMAP VOC.bmp
```

Во время выполнения варианта WM_DESTROY растровые образы удаляются из памяти:

```
case WM_DESTROY:
     /*-------------------------------------
     Удаляем объекты растровых изображений.
     -------------------------------------*/
     DeleteObject ( hBitmapS   );
     DeleteObject ( hBitmapTS  );
     DeleteObject ( hBitmapWAV );
     DeleteObject ( hBitmapVOC );

     PostQuitMessage (0);
     return 0;
```

Кроме того, перед завершением приложения мы удаляем растровые изображения из памяти, что позволяет другим программам использовать освободившуюся память. Если вы имеете плохую привычку забывать освобождать задействованную память, то пользователи впоследствии будут жаловаться на проблемы с памятью, возникающие после выполнения ваших программ.

Фон в программе FileType

В диалоговом окне **FileTypeBox** используется желтый фон. Для создания этого фона мы вначале программой Paintbrush создали файл BACKGND.bmp, сохранили его в файле типа .BMP, а затем включили этот файл в FileType.rc:

```
BkGndBitmap BITMAP BACKGND.bmp
```

В функции WinMain() мы описали указатели на фоновое изображение и кисть:

```
HBITMAP   hBitmapOfBkGnd; /* Указатель на растровое */
                          /* изображение фона.      */
HBRUSH    hBrushOfBkGnd;  /* Указатель на кисть.    */
```

Затем проинициализировали указатели на фоновое изображение и кисть следующим образом:

```
hBitmapOfBkGnd = LoadBitmap ( ghInst, "BkGndBitmap" );
hBrushOfBkGnd = CreatePatternBrush ( hBitmapOfBkGnd );
```

И наконец, инициализировали элемент hBrushOfBkGnd структуры wc:

```
wc.hbrBackground = hBrushOfBkGnd;
```

В конце функции WinMain() удаляем растровые изображения из памяти:

```
DeleteObject ( hBrushOfBkGnd );
```

Вообще-то, BACKGND.bmp, который используется в качестве фонового изображения, — это просто желтый квадрат, но описанный метод применим и для создания более сложных фоновых изображений.

Глава 6. Анимация

В настоящей главе мы рассмотрим программу, которая использует значение, возвращаемое функцией sp_PlayF(). Это значение является очень важным: оно используется многими мультимедиа-приложениями Windows.

Представленная здесь программа называется Dog. На ее примере продемонстрировано, как писать программы, отображающие движущиеся объекты и одновременно воспроизводящие звуки.

Программа Dog

Мы рекомендуем перед рассмотрением текста программы откомпилировать и запустить ее. Это поможет вам лучше разобраться в ней.

Компиляция, компоновка и запуск программы Dog

Для компиляции и компоновки программы Dog с помощью компилятора фирмы Microsoft необходимо выполнить следующие действия:

- Убедиться в том, что компьютер находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
NMAKE Dog.mak    [Enter]
```

Для компиляции и компоновки программы Dog с помощью компилятора фирмы Borland нужно выполнить такие действия:

- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
MAKE -f Dog.bmk    [Enter]
```

Для запуска программы Dog необходимо выполнить следующие действия:

- Выбрать команду **Выполнить** из меню **Файл** Диспетчера Программ Windows.
- Использовать диалоговое окно **Пролистать** для выбора файла c:\spSDK\Samp4Win\Dog.exe

Возникающее при запуске главное окно приложения приведено на рис. 6.1.

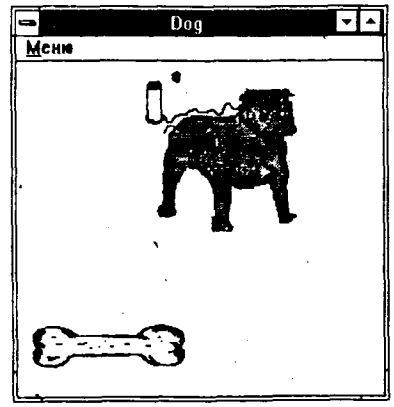


Рис. 6.1. Главное окно программы Dog.exe

Рис. 6.2. Главное меню программы Dog.exe

Главное меню программы Dog.exe (см. рис. 6.1) содержит три команды — **Завершить**, **О программе** и **Инструкции**. Если пользователь выбирает команду **Завершить**, то программа "произносит" фразу "Good-Bye" и завершается. Если он выбирает команду **О программе**, то выводится диалоговое окно **О программе Dog**, которое приведено на рис. 6.3. Выбор команды **Инструкции** приводит к появлению диалогового окна **Инструкции**, изображенного на рис. 6.4. В этом диалоговом окне выводится сообщение о том, что для того, чтобы разозлить собаку, нужно нажать на клавишу [B] или щелкнуть мышью на изображении кости. После этого собака начнет набрасываться, прыгать и лаять.

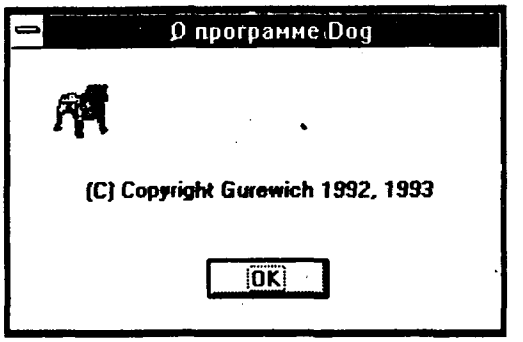


Рис. 6.3. Диалоговое окно О программе приложения Dog

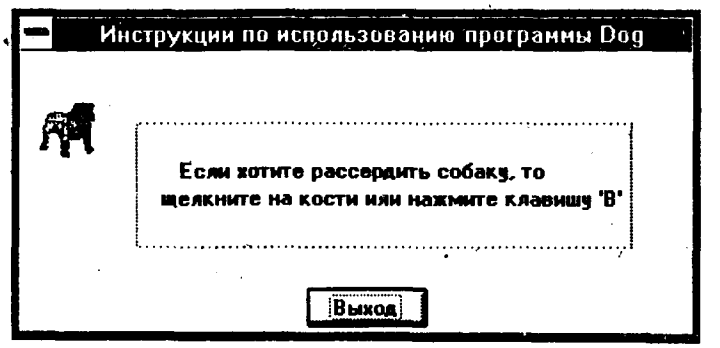


Рис. 6.4. Диалоговое команды Инструкции программы Dog

Что такое анимация?

Анимация — это процесс перемещения двухмерных изображений на фоне неподвижного изображения (так называемого "фона") с целью придания этому движению реалистичности. При анимации имитация движения достигается за счет вывода изображений со скоростью несколько кадров в секунду, благодаря чему создается эффект плавности перемещения.

Естественно, при создании анимации не нужна такая скорость отображения кадров, как в фильмах. Например, для создания эффекта нажатия командной кнопки, без которого не обходится ни одно приложение Windows, используется только два кадра (фрейма). В одном из них кнопка изображена в отжатом положении, в другом — в нажатом. Когда вы нажимаете на командную кнопку, внутренняя программа обработки событий этой кнопки сначала отображает ее в отжатом положении, а затем сразу же — в нажатом, создавая желаемый эффект нажатия на механическую кнопку.

В программе Dog используется именно такой метод. Есть всего два фрейма: на первом собака изображена с закрытой пастью и в спокойном состоянии, а на втором пасть у нее открыта, из пасти брызжет слюна, мускулы напряжены. Эти фреймы приведены на рис. 6.5. Кость, из-за которой так нервничает собака, изображена на рис. 6.6.

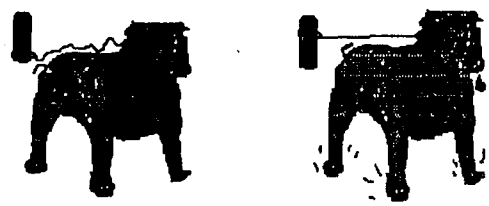


Рис. 6.5. Два фрейма, используемые в программе Dog

Рис. 6.6. Растровое изображение Bone.bmp

Создание растровых изображений

Оба фрейма и изображение кости созданы с помощью редактора Image Editor, который входит в комплект поставки SDK для Windows. Конечно, вы можете создать изображение с помощью Paintbrush или другого графического редактора, а затем через буфер обмена Windows скопировать его в Image Editor. Если вы используете Image Editor, то необходимо указать, что новое изображение будет иметь формат .BMP. Кроме того, Image Editor спрашивает о габаритах будущего изображения — они определяют размеры готового BMP-файла.

Для изображения собаки мы задали следующие размеры: 125 пикселов в высоту и 125 пикселов в ширину, что обусловило объем созданного растрового образа — 8188 байт. Для изображения Bone.bmp мы задали габариты 125 на 32 пиксела. Готовый файл с изображением кости имел размер 2166 байт.

Процесс отображения фреймов с изображением собаки очень прост. В большинстве случаев такую простую анимацию можно создать, вовсе не будучи профессиональным художником: сначала мы вывели изображение собаки с закрытой пастью, а затем заменили его изображением собаки с открытой пастью. Все, что мы сделали для создания второго изображения — это загрузили исходное, изменили положение поводка, раскрасили внутреннюю часть пасти красным цветом и подрисовали мускулы, а также капли воды для создания впечатления, что из пасти брызжет слюна, — еще бы, ведь собака такая злая.

Файлы, входящие в состав программы Dog

Полный комплект программы Dog содержит следующие файлы:

Dog.c	Текст программы на C
Dog.h	Файл .h
Dog.rc	Описание ресурсов программы
Dog.def	Файл определения модуля программы
Dog.mak	Make-файл программы
DogOpen.bmp	Изображение собаки с открытой пастью
DogClose.bmp	Изображение собаки с закрытой пастью
Bone.bmp	Изображение кости
Dog.ico	Пиктограмма программы
BckGnd.bmp	Фон, используемый для кисти

Все эти файлы находятся на дискете, прилагаемой к книге, и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Приведем полный листинг программы Dog (см. листинги 6.1–6.5).

Листинг 6.1. Dog.h

```
/*====================================
 ИМЯ ФАЙЛА: Dog.h

 (C) Copyright Gurewich 1992, 1993
 ====================================*/

/*---------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc      ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export InstDlgProc  ( HWND, UINT, UINT, LONG ) ;

void BarkingShow         ( HWND hWnd );
void PlayIt              ( HWND hWNd );
void DisplayDogWithOpen  ( HWND hWnd );
void DisplayDogWithClose ( HWND hWnd );

/*------
 #define
 -------*/
#define IDM_QUIT  1          /* Команда меню Завершить    */
#define IDM_ABOUT 2          /* Команда меню О программе  */
#define IDM_INSTRUCTIONS 3   /* Команда меню Инструкции   */

#define EXIT_PB 100          /* Командная кнопка Выход        */
                             /* в диалоговом окне Инструкции. */

/*--------------------
 Глобальные переменные
 ---------------------*/
char       gszAppName[] = "Dog" ; /* Имя нашего приложения. */
```

```
HINSTANCE  ghInst;              /* Текущий экземпляр. */
HBRUSH     ghBrushOfBkGnd;      /* Для кисти.         */
HBITMAP    ghDogWithOpenMouth;  /* Для растрового изображения собаки.*/
HBITMAP    ghDogWithCloseMouth; /* Для растрового изображения собаки.*/
```

Листинг 6.2. Dog.c

```
/*============================================================
  ПРОГРАММА: Dog.c
  ---------
  (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

  ОПИСАНИЕ ПРОГРАММЫ:
  -------------------
  Эта программа основана на Generic1.

  Если пользователь выполняет щелчок на кости, то это приводит
  несчастного пса в бешенство.

 ============================================================*/

/*--------
 #include
 --------*/
/*-----------------------------------------------
 windows.h  требуется для всех приложений Windows.
 -----------------------------------------------*/
#include <windows.h>

/*-------------------------------------------------------------
 sp4Win.h требуется для того, чтобы функции sp_,  макросы SP_ и
 #define из библиотеки звуковой поддержки могли быть использованы
 в этом приложении.
 -------------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-------------------------------------------------------------
 Определения и прототипы, специфические для данного приложения.
 -------------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\Dog.h"

/*-------------------------------------------------------------
 Для стандартных функций C, используемых в этом приложении.
 -------------------------------------------------------------*/
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>

/*==================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
```

```
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
{
HWND     hWnd; /* Идентификатор окна нашего приложения.        */
MSG      msg;  /* Сообщения, обрабатываемые нашим приложением.*/
WNDCLASS wc;   /* Класс окна для нашего приложения.            */

HBITMAP  hBitmapOfBkGnd;  /* Указатель на растровый рисунок    */
                          /* изображения фона.                 */

/*-------------------------------------------------
 Создаем экземпляр глобальной переменной hInstance.
 -------------------------------------------------*/
ghInst = hInstance;

/*-------------
 Создаем кисть.
 -------------*/
hBitmapOfBkGnd = LoadBitmap ( ghInst, "BackGround"   );
ghBrushOfBkGnd = CreatePatternBrush ( hBitmapOfBkGnd );

/*-----------------------------------------------------------
 Инициализируем структуру оконного класса и создаем  класс.
 -----------------------------------------------------------*/
if ( !hPrevInstance )
   {
   /*-------------------------------------------------
   Если условие выполняется, то это самый первый запуск
   нашего приложения.
   -------------------------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfDog" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW     ) ;
   wc.hbrBackground = ghBrushOfBkGnd; /* Кисть, которую мы создали с*/
                                      /* помощью растрового рисунка.*/
   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = gszAppName ;

   /*-----------------
   Регистрируем окно.
   -----------------*/
   RegisterClass ( &wc );

   }/* конец if(!hPrevInstance) */


/*-----------------------------
 Создаем окно нашего приложения.
 -----------------------------*/
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
```

```
                        WS_OVERLAPPEDWINDOW,
                        10,
                        10,
                        300,
                        300,
                        NULL,
                        NULL,
                        ghInst,
                        NULL );

  /*-------------------------------------------
   Отображаем и обновляем окно нашего приложения.
   -------------------------------------------*/
  ShowWindow   ( hWnd, nCmdShow );
  UpdateWindow ( hWnd );

  /*-------------------------
   Цикл обработки сообщений.
   -------------------------*/
  while ( GetMessage ( &msg, NULL, 0, 0 ) )
        {
        TranslateMessage ( &msg );
        DispatchMessage  ( &msg );
        }

  /*----------------------------------------
   Удаляем фоновый объект, который мы создали.
   ----------------------------------------*/
  DeleteObject ( hBitmapOfBkGnd );
  DeleteObject ( ghBrushOfBkGnd );

  return msg.wParam ;

  } /* Конец функции. */
  /*========================= конец WinMain() ====================*/

  /*=================
   ФУНКЦИЯ: WndProc()
   ==================*/
  /*-----------------------------
   ОПИСАНИЕ: обработка сообщений.
   -----------------------------*/

  long FAR PASCAL _export WndProc ( HWND hWnd,
                                    UINT message,
                                    UINT wParam,
                                    LONG lParam )
  {
  /*-----------------------------------
   Локальные и статические переменные.
   -----------------------------------*/
  static HDC         hdc;    /* Для вывода. */
  static PAINTSTRUCT ps;     /* Для вывода. */
  static HDC         hMemDC; /* Для вывода растрового изображения. */
  static HBITMAP     hBone; /* Для растрового изображения кости. */
```

```
static FARPROC lpfnAboutDlgProc;/* Для диалогового окна О программе. */
static FARPROC lpfnInstDlgProc ;/* Для диалогового окна Инструкции.  */

static int     iOpenBarkResult;
static int     iOpenHelloResult;

switch ( message )
       {
       case WM_CREATE:
            /*-------------------------------
            Начинаем работу со звуком (лаем).
            -------------------------------*/
            iOpenBarkResult = sp_OpenSession (
                              "c:\\spSDK\\TSfiles\\Bark.ts",
                               SP_NON_STAND_ALONE,
                               0L,
                               SP_TS_TYPE);

            /*-----------------------------------------
            Завершаем работу, если не можем открыть файл.
            -----------------------------------------*/
            if ( iOpenBarkResult != SP_NO_ERRORS )
               {
               MessageBox ( NULL,
                            "Невозможно начать работу со звуком.",
                            "Сообщение от dog.exe",
                            MB_ICONINFORMATION );

               /*-------------------------
               Завершаем наше приложение.
               -------------------------*/
               SendMessage ( hWnd,
                             WM_DESTROY,
                             0,
                             0 );
               }

            /*---------------------------------------
            Получаем lpfnAboutDlgPro, указывающее на
            диалоговое окно команды О программе.
            ---------------------------------------*/
            lpfnAboutDlgProc =
            MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);

            /*---------------------------------------
            Получаем lpfnInstDlgProc, указывающее на
            диалоговое окно Инструкции.
            ---------------------------------------*/
            lpfnInstDlgProc =
            MakeProcInstance (( FARPROC) InstDlgProc, ghInst);

            /*-------------------------------------------------
            Получаем указатели на растровое изображение собаки
            и растровое изображение кости.
            -------------------------------------------------*/
```

```
        ghDogWithCloseMouth = LoadBitmap ( ghInst,
                                           "DogCloseMouth" );

        ghDogWithOpenMouth  = LoadBitmap ( ghInst,
                                           "DogOpenMouth"  );

        hBone               = LoadBitmap ( ghInst,
                                           "Bone" );
        return 0;

    case WM_PAINT:
        /*-----------------------------
        Рисуем собаку с закрытой пастью.
        -----------------------------*/
        hdc = BeginPaint ( hWnd, &ps );
        hMemDC = CreateCompatibleDC ( hdc );
        SelectObject ( hMemDC, ghDogWithCloseMouth );
        BitBlt ( hdc,
                 100,
                 10,
                 120,
                 120,
                 hMemDC,
                 0,
                 0,
                 SRCCOPY );
        DeleteDC ( hMemDC );
        /*-----------
        Рисуем кость.
        -------------*/
        hMemDC = CreateCompatibleDC ( hdc );
        SelectObject ( hMemDC, hBone );
        BitBlt ( hdc,
                 10,
                 200,
                 125,
                 32,
                 hMemDC,
                 0,
                 0,
                 SRCCOPY );
        DeleteDC ( hMemDC );
        EndPaint ( hWnd, &ps );

        return 0;
    case WM_COMMAND:
        /*--------------------
        Обработка команд меню.
        ----------------------*/
        switch (wParam)
               {
               case IDM_QUIT:
                    /*-----------------------------------
                    Пользователь выбрал команду Завершить.
                    -----------------------------------*/
```

```
                    DestroyWindow (hWnd);
                    return 0L;

               case IDM_ABOUT :
                    /*-------------------------------------
                    Пользователь выбрал команду О программе.
                    -------------------------------------*/
                    DialogBox ( ghInst,
                                "AboutBox",
                                hWnd,
                                lpfnAboutDlgProc );
                    return 0L;

               case IDM_INSTRUCTIONS :
                    /*-------------------------------------
                    Пользователь выбрал команду Инструкции.
                    -------------------------------------*/
                    DialogBox ( ghInst,
                                "InstructionsBox",
                                hWnd,
                                lpfnInstDlgProc );
                    return 0L;

              }/* конец switch (wParam) */

case WM_CHAR:
     /*---------------------------------------
     Пользователь нажал клавишу на клавиатуре.
     ---------------------------------------*/

     /*---------------------------------
     Преобразование к верхнему регистру.
     ---------------------------------*/
     wParam = toupper ( wParam );

     switch ( wParam )
            {
            case 'B':
                 /*--------------------
                 Собака начинает лаять.
                 --------------------*/
                 BarkingShow ( hWnd );
                 return 0;
            }

case WM_LBUTTONDOWN:
     /*-------------------------------------------------
     Сообщение получено по факту нажатия пользователем
     левой кнопки мыши.
     -------------------------------------------------*/

     if (
        /*-------------------------
        Район, где находится кость.
        -------------------------*/
```

```
            LOWORD ( lParam ) < 10 +  125  &&
            LOWORD ( lParam ) > 10         &&
            HIWORD ( lParam ) < 200 + 32   &&
            HIWORD ( lParam ) > 200
            )
            {
            /*--------------------
            Собака начинает лаять.
            ----------------------*/
            BarkingShow ( hWnd );
            }
            return 0L;

    case WM_DESTROY:
         /*-------------------------------
         Открываем звуковой файл Hello.ts.
         -------------------------------*/
         iOpenHelloResult =
         sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                          SP_NON_STAND_ALONE,
                          0L,
                          SP_TS_TYPE ) ;

        if ( iOpenHelloResult == SP_NO_ERRORS )
           {
           /*-----------------------------------
           Воспроизводим часть звукового файла.
           Сколько играть: от 30000 до 40000
           Фраза: "Good-Bye"
           -----------------------------------*/
           sp_PlayF ( 30000L, 40000L ) ;
           }

           /*----------------------------
           Удаляем растровые изображения.
           ----------------------------*/
           DeleteObject ( ghDogWithCloseMouth );
           DeleteObject ( ghDogWithOpenMouth  );
           DeleteObject ( hBone               );

           PostQuitMessage (0);
           return 0;
    }/* конец switch (message) */

/*----------------------------
 Сообщение не было обработано.
 ----------------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}/* конец WndProc() */
/*=================== конец WndProc() ====================*/


/*======================
 ФУНКЦИЯ: AboutDlgProc()
 ======================*/
```

```
/*-------------------------------------------------
 ОПИСАНИЕ:
 Это функция обработки диалогового окна О программе.
 -------------------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;

       case WM_COMMAND :
            switch ( wParam )
                   {
                   case IDOK :
                   case IDCANCEL :
                        EndDialog ( hDlg, 0 );
                        return TRUE;
                   }
       }/* конец switch(message) */

return FALSE ;
}/* Конец функции. */
/*=============== конец AboutDlgProc() ==============*/


/*=====================
 ФУНКЦИЯ: InstDlgProc()
 =====================*/
/*-------------------------------------------------
 ОПИСАНИЕ:
 Это функция, реализующая диалоговое окно Инструкции.
 -------------------------------------------------*/
BOOL FAR PASCAL _export InstDlgProc ( HWND hDlg,
                                      UINT message,
                                      UINT wParam,
                                      LONG lParam )
{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;

       case WM_COMMAND :
            switch ( wParam )
                   {
                   case EXIT_PB :
                   case IDOK :
                   case IDCANCEL :
                        EndDialog ( hDlg, 0 );
                        return TRUE;
                   }
       }
```

```
return FALSE ;
}/* Конец функции. */
/*=============== конец InstDlgProc() ==============*/

/*==================
 ФУНКЦИЯ BarkingShow
 ====================*/
/*--------------------------------
 ОПИСАНИЕ:
 --------
 Функция, реализующая лай собаки.
 --------------------------------*/
void BarkingShow ( HWND hWnd )
{
int  iNumberOfPlays;

for ( iNumberOfPlays = 0; iNumberOfPlays < 4; iNumberOfPlays++)
    {
    PlayIt ( hWnd );
    }

DisplayDogWithClose ( hWnd );

}/* Конец функции. */
/*============ конец функции ==========*/


/*==============
 ФУНКЦИЯ: PlayIt
 ===============*/
/*-----------------------------------
 ОПИСАНИЕ:
 ---------
 Воспроизводит звуковой файл один раз.
 -----------------------------------*/
void PlayIt ( HWND hWnd )
{
int  iMouthFlag   = 0;
long lCurrentByte = 0L;

while (1)
      {
      lCurrentByte = sp_PlayF ( lCurrentByte,
                                lCurrentByte + 2000L );

      if ( lCurrentByte == 0L )
         {
         break;
         }

      if ( iMouthFlag == 0 )
         {
         /*--------------------------------
         Выводит собаку с открытой пастью.
         --------------------------------*/
         iMouthFlag = 1;
```

```
            DisplayDogWithOpen  ( hWnd );
            continue;
            }

        if ( iMouthFlag == 1 )
            {
            /*--------------------------------
            Выводит собаку с закрытой пастью.
            --------------------------------*/
            iMouthFlag = 0;
            DisplayDogWithClose ( hWnd );
            continue;
            }

        }/* конец цикла while(1). */

}/* Конец функции. */
/*============ конец функции ===========*/


/*============================
 ФУНКЦИЯ: DisplayDogWithOpen
 ============================*/
/*--------------------------------
 ОПИСАНИЕ:
 ---------
 Выводит собаку с открытой пастью.
 --------------------------------*/
void DisplayDogWithOpen ( HWND hWnd )
{
HDC hdc;
HDC hMemDC;

/*--------------------------------
 Рисуем собаку с открытой пастью.
 --------------------------------*/
hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghDogWithOpenMouth );

BitBlt ( hdc,
         100,
         10,
         125,
         125,
         hMemDC,
         0,
         0,
         SRCCOPY );

DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );
}
/*============ конец функции ===========*/
```

```
/*============================
 ФУНКЦИЯ: DisplayDogWithClose
 ============================*/
/*------------------------------
 ОПИСАНИЕ:
 ---------
 Рисуем собаку с закрытой пастью.
 --------------------------------*/
void DisplayDogWithClose ( HWND hWnd )
{
static HDC hdc;
static HDC hMemDC;

/*------------------------------
 Рисуем собаку с закрытой пастью.
 --------------------------------*/
hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghDogWithCloseMouth );

BitBlt ( hdc,
         100,
         10,
         125,
         125,
         hMemDC,
         0,
         0,
         SRCCOPY );

DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );
}
/*============ конец функции ===========*/
```

Листинг 6.3. Dog.rc

```
/*=============================================
 ИМЯ ФАЙЛА: Dog.rc
 ---------
 ОПИСАНИЕ ФАЙЛА:
 ---------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 =============================================*/

/*-------
 #include
 --------*/
#include <windows.h>
#include "Dog.h"
```

```
/*---
 Меню
 ----*/
Dog  MENU
BEGIN
   POPUP  "&Меню"
      BEGIN
       MENUITEM "&Завершить", IDM_QUIT
       MENUITEM "&О программе", IDM_ABOUT
       MENUITEM "&Инструкции", IDM_INSTRUCTIONS
       END
END

IconOfDog ICON Dog.ico

/*---------------------
 Растровые изображения.
 ----------------------*/
BackGround    BITMAP BackGnd.bmp
DogOpenMouth  BITMAP DogOpen.bmp
DogCloseMouth BITMAP DogClose.bmp
Bone          BITMAP Bone.bmp

/*---------------------------
 Диалоговое окно О программе Dog.
 ---------------------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "О программе Dog"
/* необходим локализованный шрифт MS Sans Serif */
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "OK", IDOK, 64, 75, 40, 14
    CTEXT           "(C) Copyright Gurewich 1992, 1993",
                    -1, 13, 47, 137, 18
    ICON            "IconOfDog", -1, 14, 12, 18, 20
END

/*-----------------------------------
 Диалоговое окно для выдачи инструкций.
 -----------------------------------*/
InstructionsBox DIALOG 41, 45, 222, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Инструкции по использованию программы Dog"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "Выход", EXIT_PB, 95, 84, 40, 14
    CTEXT           "Если хотите рассердить собаку, то",
                    -1, 37, 38, 152, 13
    CTEXT           "щелкните на кости или нажмите клавишу 'B'",
                    -1, 47, 48, 164, 8
    CONTROL         "", -1, "Static", SS_GRAYFRAME, 38, 25, 164, 45
    ICON            "IconOfDog", -1, 7, 18, 18, 20
END
```

Листинг 6.4. Dog.def

```
;==================================
; Файл определения модуля для Dog.c
;==================================

NAME          Dog

DESCRIPTION   'Программа Dog. (C) Copyright Gurewich 1992, 1993'

EXETYPE       WINDOWS

STUB          'WINSTUB.EXE'

CODE  PRELOAD MOVEABLE DISCARDABLE

DATA  PRELOAD MOVEABLE MULTIPLE

HEAPSIZE      1024
STACKSIZE     8192
```

Листинг 6.5. Dog.mak

```
#=============
# Dog.mak
#=============

Dog.exe : Dog.obj Dog.h Dog.def Dog.res
    link /nod Dog.obj, Dog.exe, NUL, \
       slibcew.lib oldnames.lib libw.lib commdlg \
       c:\spSDK\TegoWlib\TegoWin.lib, \
       Dog.def
    rc -t Dog.res

Dog.obj : Dog.c Dog.h
   cl -c -G2sw -Ow -W3 -Zp  Dog.c

Dog.res : Dog.rc Dog.h Dog.ico
   rc -r Dog.rc
```

Применение значения, возвращаемого функцией sp_PlayF()

Функция sp_PlayF() возвращает длинное целое. Это длинное целое определяет последний байт, воспроизведенный TSEngine. Так, вызов

```
sp_PlayF ( 0L, 3000L );
```

возвращает 3000L, поскольку последним воспроизведенным байтом был байт со смещением 3000.

Впрочем, иногда координаты, возвращаемые функцией sp_PlayF(), не соответствуют реальным. Например, если длина воспроизводимого файла — 40000 байт, а вы в качестве второго параметра функции sp_PlayF() передаете значение 50000L:

```
sp_PlayF ( 0L, 50000L );
```

то TSEngine воспроизведет байты с 0 по 40000 и возвратит 0; это будет означать, что достигнут конец звукового файла.

Если передаваемые параметры заведомо неверны (например, начало воспроизведения задано с 30000L, а конец — 20000L), то функция возвращает 0 и никаких действий при этом не производит (то есть выполнение программы не будет прервано).

Примечание:

Значение, возвращаемое функцией sp_PlayF(), является длинным целым, указывающим на последний байт, воспроизведенный TSEngine.

Если второй параметр, передаваемый функции sp_PlayF(), больше размера файла, то функция sp_PlayF() возвратит 0L. И в этом случае TSEngine воспроизведет все байты, вплоть до самого конца.

Если координаты, передаваемые функции sp_PlayF(), неверны и TSEngine не может ничего воспроизвести, то функция sp_PlayF() вернет 0L.

Функция WinMain() программы Dog.c

Функция WinMain() программы Dog.c загружает растровые изображения фона и создает кисть:

```
hBitmapOfBkGnd = LoadBitmap ( ghInst, "BackGround"   );
ghBrushOfBkGnd = CreatePatternBrush ( hBitmapOfBkGnd );
```

Затем элемент hbrBackground структуры wc инициализируется значением ghBrushOfBkGnd:

```
wc.hbrBackground = ghBrushOfBkGnd;
```

Функция CreateWindow() определяет главное окно как окно, верхний левый угол которого имеет пиксельные координаты (10,10), что задается четвертым и пятым параметрами функции CreateWindow(). Ширина и высота окна задается пиксельными размерами 300 и 300 соответственно, что определяется шестым и седьмым параметрами функции CreateWindow():

```
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      10,
                      10,
                      300,
                      300,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );
```

Функция WndProc() программы Dog.c

Функция WndProc() обрабатывает варианты, соответствующие сообщениям, которые поступают в процессе выполнения программы. Рассмотрим эти варианты.

Вариант WM_CREATE

В варианте WM_CREATE мы начинаем сеанс работы со звуковым файлом типа TS Bark.ts, находящимся в директории c:\spSDK\TSfiles\. Если сеанс не был успешно открыт, то программа завершается:

```
/*-------------------------------
Начинаем работу со звуком (лаем).
-------------------------------*/
iOpenBarkResult = sp_OpenSession (
                  "c:\\spSDK\\TSfiles\\Bark.ts",
                  SP_NON_STAND_ALONE,
                  0L,
                  SP_TS_TYPE);

/*-----------------------------------------
Завершаем работу, если не можем открыть файл.
-----------------------------------------*/
if ( iOpenBarkResult != SP_NO_ERRORS )
   {
   MessageBox ( NULL,
                "Невозможно начать работу со звуком.",
                "Сообщение от dog.exe",
                MB_ICONINFORMATION );

  /*------------------------
  Завершаем наше приложение.
  ------------------------*/
  SendMessage ( hWnd,
                WM_DESTROY,
                0,
                0 );
   }
```

Затем получаем указатели типа lpfn на диалоговые окна **О программе Dog** и **Инструкции**:

```
lpfnAboutDlgProc =
MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);

lpfnInstDlgProc =
MakeProcInstance (( FARPROC) InstDlgProc, ghInst);
```

После этого загружаем растровые изображения собаки с открытой и закрытой пастью, а также растровое изображение кости:

```
ghDogWithCloseMouth = LoadBitmap ( ghInst, "DogCloseMouth" );
ghDogWithOpenMouth  = LoadBitmap ( ghInst, "DogOpenMouth"  );
hBone               = LoadBitmap ( ghInst, "Bone" );
```

Вариант WM_PAINT

При поступлении сообщения WM_PAINT выводим изображения собаки с закрытой пастью и кости. Для вывода этих растровых изображений мы используем функцию BitBlt():

```
hdc = BeginPaint ( hWnd, &ps );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghDogWithCloseMouth );
BitBlt ( hdc,
         100,
         10,
         120,
         120,
         hMemDC,
         0,
         0,
         SRCCOPY );
DeleteDC ( hMemDC );
```

```
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, hBone );
BitBlt ( hdc,
         10,
         200,
         125,
         32,
         hMemDC,
         0,
         0,
         SRCCOPY );
         DeleteDC ( hMemDC );
         EndPaint ( hWnd, &ps );
```

Вариант WM_DESTROY

При поступлении сообщения WM_DESTROY открываем сеанс работы со звуковым файлом c:\spSDK\TSfiles\Hello.ts и воспроизводим участок звукового файла, соответствующий фразе "Good-Bye". Если сеанс не открывается, то воспроизведение игнорируется:

```
/*---------------------------------
Открываем звуковой файл Hello.ts.
---------------------------------*/
iOpenHelloResult =
sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                 SP_NON_STAND_ALONE,
                 0L,
                 SP_TS_TYPE ) ;

if ( iOpenHelloResult == SP_NO_ERRORS )
   {
   /*-----------------------------------
   Воспроизводим часть звукового файла.
   Сколько играть: от 30000 до 40000
   Фраза: "Good-Bye"
   -----------------------------------*/
   sp_PlayF ( 30000L, 40000L ) ;
   }
```

В последней части варианта WM_DESTROY удаляем из памяти объекты, которые были созданы при обработке варианта WM_CREATE:

```
DeleteObject ( ghDogWithCloseMouth );
DeleteObject ( ghDogWithOpenMouth  );
DeleteObject ( hBone               );
```

Обратите внимание на то, что в варианте WM_CREATE мы проверяем, корректно ли был открыт сеанс работы со звуковым файлом Bark.ts. Если нет, то завершаем программу посылкой сообщения WM_DESTROY:

```
SendMessage ( hWnd,
              WM_DESTROY,
              0,
              0 );
```

Вариант WM_CHAR

Сообщение WM_CHAR посылается всякий раз, когда пользователь нажимает клавишу на клавиатуре. При поступлении этого сообщения мы, прежде всего, преобразуем полученный символ в соответствующий символ верхнего регистра, после чего проверяем, является ли полученный символ символом 'B'. Если да, то вызываем функцию BarkinShow():

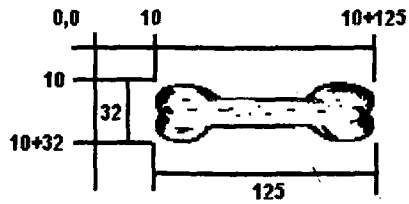
```
wParam = toupper ( wParam );

switch ( wParam )
       {
       case 'B':
            /*--------------------
            Собака начинает лаять.
            ----------------------*/
            BarkingShow ( hWnd );
            return 0;
       }
```

Вариант WM_LBUTTONDOWN

Сообщение WM_LBUTTONDOWN поступает всякий раз, когда пользователь нажимает левую кнопку мыши. lParam в этом сообщении представляет собой координаты *x* и *y* курсора мыши в момент нажатия левой кнопки: младший байт lParam — это координата *x*, а старший — координата *y*. При поступлении сообщения WM_LBUTTONDOWN проверяем, не находится ли курсор мыши в том прямоугольнике, в котором расположена кость. Поскольку в варианте WM_CREATE мы вывели растровое изображение кости начиная с координат (10,10), а размеры изображения кости — 32 на 125 пикселов (см. рис. 6.7), то следующим выражением проверяем, выполнил ли пользователь щелчок на кости.

Рис. 6.7. Начальные координаты и размеры изображения кости




```
if (
   /*---------------------------------
   Область, в которой находится кость.
   -----------------------------------*/
   LOWORD ( lParam ) < 10 +  125  &&
   LOWORD ( lParam ) > 10         &&
   HIWORD ( lParam ) < 200 + 32   &&
   HIWORD ( lParam ) > 200
   )
   {
   /*--------------------
   Собака начинает лаять.
   ----------------------*/
   BarkingShow ( hWnd );
   }
   return 0L;
```

Функция BarkingShow()

Функция BarkingShow() воспроизводит лай, а также выводит изображения собаки с открытой и закрытой пастью. Это делается с помощью цикла

```
for (iNumberOfPlays = 0; iNumberOfPlays <= 4; iNumberOfPlays++ )
{
PlayIt(hWnd);
}
```

В результате выполнения данного цикла просто четыре раза вызывается функция PlayIt(). После этого наступает очередь функции DisplayDogWithClose():

```
DisplayDogWithClose(hWnd);
```

Функция DisplayDogWithClose() просто выводит изображение собаки с закрытой пастью, поскольку по окончании лая собака, естественно, должна закрыть пасть.

Функция PlayIt()

В функции PlayIt() в качестве основного компонента используется цикл while(1):

```
long lCurrentByte = 0L;

while (1)
      {
      lCurrentByte = sp_PlayF ( lCurrentByte,
                                lCurrentByte + 2000L );

      if ( lCurrentByte == 0L )
         {
         break;
         }
      .................................
      ........ Анимация................
      .................................
      }
```

В теле цикла while(1) воспроизводится звуковой файл начиная с позиции 0L. Файл воспроизводится фрагментами по 2000 байт. Приведем описание этого цикла.

Напомним, что функция sp_PlayF() возвращает количество воспроизведенных байтов. Это означает, что переменной lCurrentByte после первого вызова функции sp_PlayF() будет присвоено значение 2000L.

Затем воспроизводятся еще 2000 байтов от позиции 2000 до позиции 4000.

Этот процесс продолжается вплоть до завершения воспроизводимого файла.

Естественно, что в какой-то момент времени второй аргумент, передаваемый функции sp_PlayF(), превысит размер воспроизводимого файла. В таком случае, как было описано ранее, TSEngine воспроизведет файл до конца и вернет 0L.

Цикл while(1) прерывается при получении от функции sp_PlayF() значения 0L:

```
if ( lCurrentByte == 0L )
   {
   break;  /* прерываем цикл while(1) */
   }
```

Итак, анимация осуществляется после воспроизведения каждого фрагмента размером в 2000 байт. Когда пользователь выполняет программу, у него, естественно, должно создаваться впечатление, что собака открывает пасть и лает одновременно. Однако мы

знаем, что у персонального компьютера только один центральный процессор и, следовательно, он может выполнять только одну задачу в единицу времени. Чередование анимации и воспроизведения фрагментов звукового файла позволяет достичь максимального эффекта совмещения во времени указанных событий. Когда собака на экране открывает пасть, практически одновременно раздается лай.

Является ли размер фрагмента в 2000 байт удачным выбором? Давайте произведем некоторые расчеты.

Файл Bark.ts был записан с частотой дискретизации, равной 8000 Гц. Если применить к файлу типа TS формулу

$$\text{Количество секунд} = \frac{\text{Количество байтов}}{\text{Частота дискретизации}}$$

то количество секунд, которое потребуется для воспроизведения 2000 байт, будет равно

$$\text{Количество секунд} = \frac{2000}{8000} = 0{,}25 \text{ с} = 250 \text{ мс}$$

Отметим, что процесс отображения занимает менее 1 мс; следовательно, и процесс вывода изображения, и процесс воспроизведения звука для органов зрения и слуха человека осуществляются одновременно.

Важно запомнить, что операторы, реализующие анимацию, выполняются в реальном масштабе времени — в промежутках между воспроизведением фрагментов звукового файла. И поэтому этот фрагмент программы должен быть максимально эффективен. Чем меньше времени пройдет между воспроизведением звуковых фрагментов, тем лучше.

Ниже приведен фрагмент программы Dog, в котором реализована анимация:

```
if ( iMouthFlag == 0 )
   {
   /*------------------------------
   Выводит собаку с открытой пастью.
   ------------------------------*/
   iMouthFlag = 1;
   DisplayDogWithOpen  ( hWnd );
   continue;
   }

if ( iMouthFlag == 1 )
   {
   /*------------------------------
   Выводит собаку с закрытой пастью.
   ------------------------------*/
   iMouthFlag = 0;
   DisplayDogWithClose ( hWnd );
   continue;
   }
```

У нас есть флаг, который называется iMouthFlag. Этот флаг показывает текущее состояние изображения собаки: значение 1 соответствует закрытой пасти, а значение 0 — открытой.

При первой итерации цикла while(1) значение iMouthFlag равно 0. Мы проверяем это оператором if. Затем мы изменяем значение флага на 1 и выводим изображение собаки с открытой пастью при помощи вызова функции DisplayDogWithOpen():

```
DisplayDogWithOpen ( hWnd );
```

После этого используем оператор continue для возврата к началу цикла while(1).

Тело цикла while(1) выполняется вновь — воспроизводятся еще 2000 байтов, а поскольку на этот раз флаг установлен в 1, то удовлетворяется условие второго оператора if и программа вновь устанавливает флаг в 0, выводит изображение собаки с закрытой пастью и возвращает оператором continue управление в начало цикла while(1).

Описанный процесс повторяется до тех пор, пока функция sp_PlayF() не возвратит 0L. Это означает, что звуковой файл воспроизведен полностью.

Функции DisplayDogWithClose() и DisplayDogWithOpen()

Функции DisplayDogWithClose() и DisplayDogWithOpen() реализуют вывод изображений собаки соответственно с закрытой и открытой пастью. Для вывода соответствующего растрового изображения в этих функциях используется функция BitBlt().

Программа HearMe

Программа HearMe — это достаточно простая программа, окно которой содержит командную кнопку и изображение головы человека. Если пользователь нажимает командную кнопку (или клавишу [H]), то программа воспроизводит фразу "Press any key to continue" (для продолжения нажмите любую клавишу). При этом рот человека движется — и создается впечатление, что эту фразу произносит именно он.

Данную программу можно перенастроить на воспроизведение любого звукового файла. Обычно при распространении программ поставляются файлы README.TXT; однако вы можете в комплект поставки своей программы включить файл HEARME. Он, несомненно, привлечет намного больше внимания, чем файл README.

Файлы, входящие в состав программы HearMe

Полный комплект программы HearMe содержит следующие файлы:

HearMe.c	Текст программы на C
HearMe.h	Файл .h.
HearMe.rc	Описание ресурсов программы
HearMe.def	Файл определения модуля программы
HearMe.mak	Make-файл программы
FaceOpen.bmp	Изображение человека с открытым ртом
FaceClose.bmp	Изображение человека с закрытым ртом
FaceMid.bmp	Изображение человека с полуоткрытым ртом
Face.ico	Пиктограмма программы

Все эти файлы находятся на прилагаемой к книге дискете и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Для более полного понимания работы программы рекомендуем, перед тем как приступить к рассмотрению исходных текстов, откомпилировать, скомпоновать и выполнить программу HearMe.

Компиляция, компоновка и выполнение программы HearMe

Для компиляции и компоновки программы HearMe с помощью компилятора фирмы Microsoft необходимо выполнить следующие действия:

- Убедиться в том, что компьютер находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
NMAKE HearMe.mak    [Enter]
```

Для компиляции и компоновки программы HearMe с помощью компилятора фирмы Borland нужно выполнить такие действия:

- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
MAKE -f HearMe.bmk    [Enter]
```

Для запуска программы HearMe необходимо:

- Выбрать из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- Использовать диалоговое окно **Пролистать** для выбора файла

```
c:\spSDK\Samp4Win\HearMe.exe
```

Главное окно, которое появляется при запуске приложения, приведено на рис. 6.8, главное меню приложения — на рис. 6.9.

Рис. 6.8. Главное окно приложения HearMe

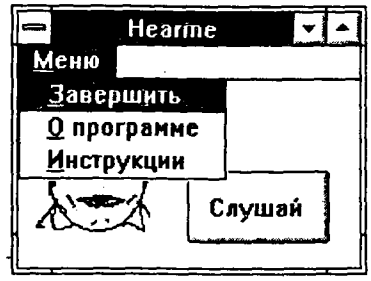


Рис. 6.9. Главное меню приложения HearMe

Если пользователь выбирает команду **О программе**, то выводится диалоговое окно **О программе HearMe**, которое приведено на рис. 6.10. Если он выбирает команду **Инструкции**, то выводится диалоговое окно **Инструкции**, приведенное на рис. 6.11.

Если пользователь выполняет щелчок на командной кнопке **Слушай**, расположенной в главном окне, то изображенный в этом окне человек произносит: "Press any key to continue". Во время воспроизведения этой фразы человек открывает и закрывает рот, а также подмигивает.

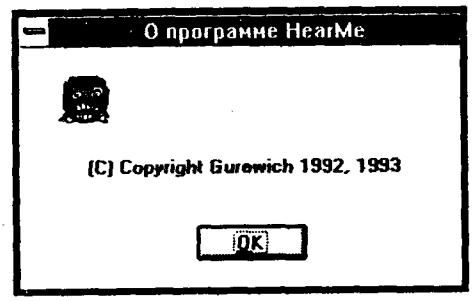


Рис. 6.10. Диалоговое окно О программе приложения HearMe

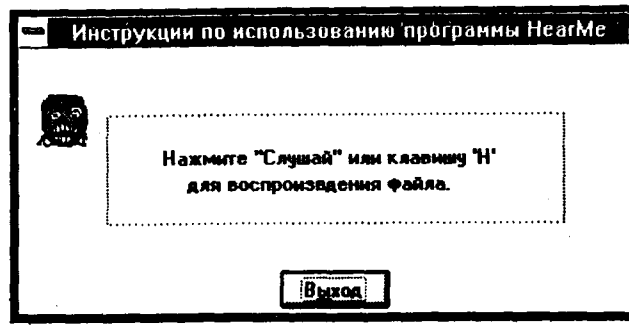


Рис. 6.11. Диалоговое окно Инструкции программы HearMe

Ниже приведен полный листинг программы HearMe (см. листинги 6.6–6.10).

Листинг 6.6. HearMe.h

```
/*================================================
 ИМЯ ФАЙЛА: Hearme.h

 (C) Copyright Gurewich 1992, 1993
 ================================================*/

/*--------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc      ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export InstDlgProc  ( HWND, UINT, UINT, LONG ) ;

void SpeakShow            ( HWND hWnd );
void DisplayFaceWithOpen  ( HWND hWnd );
void DisplayFaceWithClose ( HWND hWnd );
void DisplayFaceWithMid   ( HWND hWnd );

/*--------
 #define
 -------*/
#define IDM_QUIT         1   /* Команда меню Завершить.  */
#define IDM_ABOUT        2   /* Команда меню О программе.*/
#define IDM_INSTRUCTIONS 3   /* Команда меню Инструкции. */
```

```
#define EXIT_PB   100 /* Кнопка Выход в диалоговом окне. */
#define HEARME_PB 101 /* Кнопка Слушай в главном окне.   */

/*--------------------
 Глобальные переменные
 --------------------*/
char      gszAppName[] = "Hearme" ; /* Имя нашего приложения.*/
HINSTANCE ghInst;                   /* Текущий экземпляр.    */
HBRUSH    ghBrushOfBkGnd;           /* Для кисти.            */
HBITMAP   ghFaceWithOpenMouth; /* Для растрового изображения FaceOpen. */
HBITMAP   ghFaceWithCloseMouth;/* Для растрового изображения FaceClose.*/
HBITMAP   ghFaceWithMidMouth;  /* Для растрового изображения FaceMid.  */
```

Листинг 6.7. HearMe.c

```
/*============================================================
  ПРОГРАММА: HearMe.c
  ---------
  (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

  ОПИСАНИЕ ПРОГРАММЫ:
  -------------------
  Эта программа основана на Generic 1.

  Воспроизведение звукового файла на фоне видео.

 ============================================================*/

/*--------
 #include
 --------*/
/*-----------------------------------------------
 windows.h требуется для всех приложений Windows.
 -----------------------------------------------*/
#include <windows.h>


/*-----------------------------------------------------------------
 sp4Win.h требуется для того, чтобы функции sp_, макросы SP_ и
 #define из библиотеки звуковой поддержки могли быть использованы
 в этом приложении.
 -----------------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-------------------------------------------------------------
 Определения и прототипы, специфические для данного приложения.
 -------------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\Hearme.h"

/*-------------------------------------------------------------
 Для функций C, которым требуются стандартные #include-файлы C.
 -------------------------------------------------------------*/
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
```

```
/*==================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HWND     hWnd; /* Идентификатор окна нашего приложения.         */
MSG      msg;  /* Сообщения, обрабатываемые нашим приложением. */
WNDCLASS wc;   /* Класс окна нашего приложения.                 */

HBITMAP  hBitmapOfBkGnd; /* Указатель на растровое изображение фона. */

/*-------------------------------------------------
 Создаем экземпляр глобальной переменной hInstance.
 -------------------------------------------------*/
ghInst = hInstance;

/*-------------
 Создаем кисть.
 -------------*/
hBitmapOfBkGnd = LoadBitmap ( ghInst, "BackGround"   );
ghBrushOfBkGnd = CreatePatternBrush ( hBitmapOfBkGnd );

/*-----------------------------------------------------------
 Инициализируем структуру оконного класса и создаем класс.
 -----------------------------------------------------------*/
if ( !hPrevInstance )
   {
   /*-------------------------------------------
   Если условие выполняется, то это самый первый
   запуск нашего приложения.
   -------------------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfFace" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW      ) ;
   wc.hbrBackground = ghBrushOfBkGnd; /* Кисть, которую мы создали с*/
                                      /* помощью растрового изображения.*/
   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = gszAppName ;

   /*----------------
   Регистрируем окно.
   ----------------*/
   RegisterClass ( &wc );

   }/* конец if(!hPrevInstance) */
```

```
/*-------------------------------
 Создаем окно нашего приложения.
 -------------------------------*/
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      50,
                      50,
                      50+150,
                      50+100,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );

/*-----------------------------------------------
 Отображаем и обновляем окно нашего приложения.
 -----------------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );


/*-------------------------
 Цикл обработки сообщений.
 -------------------------*/
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage  ( &msg );
      }

/*-----------------------------------------
 Удаляем фоновый объект, который мы создали.
 -----------------------------------------*/
DeleteObject ( hBitmapOfBkGnd );
DeleteObject ( ghBrushOfBkGnd );

return msg.wParam ;

} /* Конец функции. */
/*======================== конец WinMain() ====================*/


/*=================
 ФУНКЦИЯ: WndProc()
 =================*/
/*-----------------------------
 ОПИСАНИЕ: обработка сообщений.
 -----------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
```

```
static HDC         hdc;      /* Для вывода. */
static PAINTSTRUCT ps;       /* Для вывода. */
static HDC         hMemDC;   /* Для вывода растрового изображения.   */
static FARPROC lpfnAboutDlgProc;/* Для диалогового окна О программе.*/
static FARPROC lpfnInstDlgProc ;/* Для диалогового окна Инструкции. */

int    iOpenFaceResult;
int    iOpenHelloResult;

static HWND    hButton;

switch ( message )
       {
       case WM_CREATE:
            /*-------------------------
            Начинаем работу со звуком.
            -------------------------*/
            iOpenFaceResult = sp_OpenSession (
                              "c:\\spSDK\\Sfiles\\Press.s",
                               SP_NON_STAND_ALONE,
                               0L,
                               SP_S_TYPE);
            /*-----------------------------------------
            Завершаем работу, если не можем открыть файл.
            -----------------------------------------*/
            if ( iOpenFaceResult != SP_NO_ERRORS )
               {
               MessageBox ( NULL,
                            "Невозможно начать работу со звуком.",
                            "Сообщение от hearme.exe"
                            MB_ICONINFORMATION );

            /*-------------------------
            Завершаем наше приложение.
            -------------------------*/
            SendMessage ( hWnd, WM_DESTROY, 0, 0 );
            }

       /*--------------------------------------
       Получаем lpfnAboutDlgPro, указывающее на
       диалоговое окно команды О программе.
       --------------------------------------*/
       lpfnAboutDlgProc =
       MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);
       /*--------------------------------------
       Получаем lpfnInstDlgProc, указывающее на
       диалоговое окно Инструкции.
       --------------------------------------*/
       lpfnInstDlgProc =
       MakeProcInstance (( FARPROC) InstDlgProc, ghInst);

       /*----------------------------------------
       Получаем-указатели на растровые изображения
       рисунков, отражающих выражение лица.
       ----------------------------------------*/
```

```
ghFaceWithOpenMouth  = LoadBitmap ( ghInst, "FaceOpen"  );
ghFaceWithCloseMouth = LoadBitmap ( ghInst, "FaceClose" );
ghFaceWithMidMouth   = LoadBitmap ( ghInst, "FaceMid"   );

/*--------------------
Создаем кнопку Слушай.
----------------------*/
hButton = CreateWindow ( "BUTTON",
                         "Слушай",
                         WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
                         10 + 85,
                         50,
                         80,
                         40,
                         hWnd,
                         HEARME_PB,
                         ghInst,
                         NULL);

    ShowWindow ( hButton, SW_SHOW );

    return 0;


case WM_PAINT:
     /*--------------------------
     Рисуем лицо с закрытым ртом.
     ----------------------------*/
     hdc = BeginPaint ( hWnd, &ps );
     hMemDC = CreateCompatibleDC ( hdc );
     SelectObject ( hMemDC, ghFaceWithCloseMouth );
     BitBlt ( hdc,
              10,
              20,
              10+64,
              10+64,
              hMemDC,
              0,
              0,
              SRCCOPY );
     DeleteDC ( hMemDC );
     EndPaint ( hWnd, &ps );

     return 0;


case WM_COMMAND:
     /*--------------------
     Обработка команд меню.
     ----------*-----------*/
     switch (wParam)
            {
            case HEARME_PB:
                 SetFocus  ( hWnd );
                 SpeakShow ( hWnd );
                 return 0;
```

```
                case IDM_QUIT:
                     /*-----------------------------------
                     Пользователь выбрал команду Завершить.
                     -----------------------------------*/
                     DestroyWindow (hWnd);
                     return 0L;

                case IDM_ABOUT :
                     /*-----------------------------------
                     Пользователь выбрал команду О программе.
                     -----------------------------------*/

                     DialogBox ( ghInst,
                                 "AboutBox",
                                 hWnd,
                                 lpfnAboutDlgProc );
                     return 0L;

                case IDM_INSTRUCTIONS :
                     /*-----------------------------------
                     Пользователь выбрал команду Инструкции.
                     -----------------------------------*/
                     DialogBox ( ghInst,
                                 "InstructionsBox",
                                 hWnd,
                                 lpfnInstDlgProc );
                     return 0L;

                }/* конец switch (wParam) */
          case WM_CHAR:
                /*-----------------------------------
                Пользователь нажал клавишу на клавиатуре
                -----------------------------------*/

                /*-----------------------------
                Преобразование в верхний регистр.
                -----------------------------*/
                wParam = toupper ( wParam );

                switch ( wParam )
                       {
                       case 'H':
                            /*-----------------------
                            Лицо начинает "говорить".
                            -----------------------*/
                            SpeakShow ( hWnd );
                            return 0;

                       }
                return 0L;

          case WM_DESTROY:
                /*-----------------------------
                Открываем звуковой файл Hello.ts.
                -----------------------------*/
                iOpenHelloResult =
```

```
            sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                             SP_NON_STAND_ALONE,
                             0L,
                             SP_TS_TYPE ) ;

            if ( iOpenHelloResult == SP_NO_ERRORS )
               {
               /*---------------------------------
               Воспроизводим часть звукового файла.
               Сколько играть: от 30000 до 40000
               Фраза: "Good-Bye"
               ---------------------------------*/
               sp_PlayF ( 30000L, 40000L ) ;
                        }

            /*---------------------------
            Удаляем растровые изображения.
            ---------------------------*/
            DeleteObject ( ghFaceWithCloseMouth );
            DeleteObject ( ghFaceWithMidMouth   );
            DeleteObject ( ghFaceWithOpenMouth  );

            PostQuitMessage (0);
            return 0;
        }/* конец switch (message) */

/*---------------------------
 Сообщение не было обработано.
 ---------------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}/* конец WndProc() */
/*================== конец WndProc() ==================*/

/*======================
 ФУНКЦИЯ: AboutDlgProc()
 ======================*/
/*-------------------------------------------------------------
 ОПИСАНИЕ:
 Это функция обработки диалогового окна О программе HearMe.
 -------------------------------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
        {
        case WM_INITDIALOG :
             return TRUE;

        case WM_COMMAND :
             switch ( wParam )
                    {
                    case IDOK :
```

```
                    case IDCANCEL :
                         EndDialog ( hDlg, 0 );
                         return TRUE;
                    }
        }/* конец switch(message) */
return FALSE ;
}/* Конец функции. */
/*=============== конец AboutDlgProc() ==============*/

/*======================
 ФУНКЦИЯ: InstDlgProc()
 ======================*/
/*-------------------------------------------------
 ОПИСАНИЕ:
 Это функция, реализующая диалоговое окно Инструкции.
 -------------------------------------------------*/
BOOL FAR PASCAL _export InstDlgProc ( HWND hDlg,
                                      UINT message,
                                      UINT wParam,
                                      LONG lParam )
{
switch ( message )
        {
        case WM_INITDIALOG :
             return TRUE;

        case WM_COMMAND :
             switch ( wParam )
                    {
                    case EXIT_PB :
                    case IDOK :
                    case IDCANCEL :
                         EndDialog ( hDlg, 0 );
                         return TRUE;
                    }
        }
return FALSE ;

}/* Конец функции. */
/*=============== конец InstDlgProc() ==============*/

/*=======
 ФУНКЦИЯ
 ========*/
/*-------------------------
 ОПИСАНИЕ:
 --------
 Реализует "говорящее" лицо.
 -------------------------*/
void SpeakShow ( HWND hWnd )
{
int  iMouthFlag   = 0;
long lCurrentByte = 0L;
```

```
while (1)
      {
      lCurrentByte = sp_PlayF ( lCurrentByte,
                                lCurrentByte + 2000L );

      if ( lCurrentByte == 0L )
         {
         break;
         }

      if ( iMouthFlag == 0 )
         {
         /*--------------------------
         Рисуем лицо с открытым ртом.
         ----------------------------*/
         iMouthFlag = 1;
         DisplayFaceWithOpen  ( hWnd );
         continue;
         }

      if ( iMouthFlag == 1 )
         {
         /*---------------------------
         Рисуем лицо с закрытым ртом.
         ----------------------------*/
         iMouthFlag = 2;
         DisplayFaceWithClose ( hWnd );
         continue;
         }

      if ( iMouthFlag == 2 )
         {
         /*-------------------------------
         Рисуем лицо с полуоткрытым ртом.
         -------------------------------*/
         iMouthFlag = 0;
         DisplayFaceWithMid ( hWnd );
         continue;
         }

      }/* конец цикла while() */

DisplayFaceWithClose ( hWnd );

}/* Конец функции. */
/*============ конец функции ===========*/


/*=======
 ФУНКЦИЯ
 ========*/
/*---------------------------
 ОПИСАНИЕ:
 -----------
 Рисуем лицо с открытым ртом.
 ----------------------------*/
```

```
void DisplayFaceWithOpen ( HWND hWnd )
{
HDC hdc;
HDC hMemDC;

/*----------------------------
 Рисуем лицо с открытым ртом.
 ----------------------------*/
hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghFaceWithOpenMouth );
BitBlt ( hdc,
         10,
         20,
         10+64,
         10+64,
         hMemDC,
         0,
         0,
         SRCCOPY );

DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );
}/* Конец функции. */
/*============ конец функции ===========*/


/*=======
 ФУНКЦИЯ
 ========*/
/*----------------------------
 ОПИСАНИЕ:
 ---------
 Рисуем лицо с закрытым ртом.
 ----------------------------*/
void DisplayFaceWithClose ( HWND hWnd )
{
static HDC hdc;
static HDC hMemDC;

/*----------------------------
 Рисуем лицо с закрытым ртом.
 ----------------------------*/
hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghFaceWithCloseMouth );

BitBlt ( hdc,
         10,
         20,
         10+64,
         10+64,
         hMemDC,
         0,
         0,
         SRCCOPY );
```

```
DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );
}/* Конец функции. */
/*============ конец функции ===========*/

/*=======
 ФУНКЦИЯ
 ========*/

/*--------------------------------
 ОПИСАНИЕ:
 ---------

 Рисуем лицо с полуоткрытым ртом.
 --------------------------------*/
void DisplayFaceWithMid ( HWND hWnd )
{
HDC hdc;
HDC hMemDC;

/*--------------------------------
 Рисуем лицо с полуоткрытым ртом.
 --------------------------------*/
hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghFaceWithMidMouth );

BitBlt ( hdc,
         10,
         20,
         10+64,
         10+64,
         hMemDC,
         0,
         0,
         SRCCOPY );

DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );
}
/*============ конец функции ===========*/
```

Листинг 6.8. HearMe.rc

```
/*================================================
 ИМЯ ФАЙЛА: hearme.rc
 ---------
 ОПИСАНИЕ ФАЙЛА:
 ---------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 ================================================*/
```

```
/*-------
 #include
 --------*/
#include <windows.h>
#include "Hearme.h"

/*---
 Меню
 ----*/
HearMe  MENU
BEGIN
   POPUP  "&Меню"
        BEGIN
        MENUITEM "&Завершить", IDM_QUIT
        MENUITEM "&О программе", IDM_ABOUT
        MENUITEM "&Инструкции", IDM_INSTRUCTIONS
        END
END

/*-----------------------------------------
 Определение пиктограммы с изображением лица.
 Имя файла: Face.ico
 Имя значка: IconOfFace
 -----------------------------------------*/
IconOfFace ICON Face.ico

/*---------------------
 Растровые изображения.
 ---------------------*/
FaceOpen       BITMAP FaceOpen.bmp
FaceClose      BITMAP FaceClos.bmp
FaceMid        BITMAP FaceMid.bmp
BackGround     BITMAP BackGnd.bmp

/*----------------------------------
 Диалоговое окно О программе HearMe.
 ----------------------------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "О программе HearMe"
/* необходим локализованный шрифт MS Sans Serif */
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON        "&OK", IDOK, 64, 75, 40, 14
    CTEXT             "(C) Copyright Gurewich 1992, 1993",
                      -1, 13, 47, 137, 18
    ICON              "IconOfFace", -1, 14, 12, 18, 20
END

/*-------------------------------------
 Диалоговое окно для выдачи инструкций.
 -------------------------------------*/
InstructionsBox DIALOG 41, 45, 222, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Инструкции по использованию программы HearMe"
```

```
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "&Выход", EXIT_PB, 95, 84, 40, 14
    CTEXT           "Нажмите ""Слушай"" или клавишу 'H'",
                    -1, 37, 38, 152, 13
    CTEXT           "для воспроизведения файла.", -1, 47, 48, 125, 8
    CONTROL         "", -1, "Static", SS_GRAYFRAME, 32, 26, 166, 41
    ICON            "IconOfFace", -1, 7, 18, 18, 20
END
```

Листинг 6.9. HearMe.def

```
;======================================
; Файл определения модуля для HearMe.c
;======================================

NAME            HearMe

DESCRIPTION     'Программа HearMe. (C) Copyright Gurewich 1992, 1993'

EXETYPE         WINDOWS

STUB            'WINSTUB.EXE'

CODE  PRELOAD   MOVEABLE DISCARDABLE

DATA  PRELOAD   MOVEABLE MULTIPLE

HEAPSIZE       1024
STACKSIZE      8192
```

Листинг 6.10. HearMe.mak

```
#=============
# Hearme.mak
#=============

Hearme.exe : Hearme.obj Hearme,h Hearme.def Hearme.res
    link /nod Hearme.obj, Hearme.exe, NUL, \
       slibcew.lib oldnames.lib libw.lib commdlg \
       c:\spSDK\TegoWlib\TegoWin.lib, \
       Hearme.def
    rc -t Hearme.res

Hearme.obj : Hearme.c Hearme.h
   cl -c -G2sw -Ow -W3 -Zp  Hearme.c

Hearme.res : Hearme.rc Hearme.h Face.ico
   rc -r Hearme.rc
```

Воспроизведение фразы "Good-Bye" и завершение программы

Если пользователь выбирает из системного меню команду **Закрыть** или из главного меню программы команду **Завершить**, то программа воспроизводит фразу "Good-Bye" и завершает свою работу. Обратите внимание на то, что мы воспроизводим "Good-Bye" только в том случае, если сеанс работы с файлом c:\spSDK\TSfiles\Hello.ts был открыт успешно.

Изображение лица в программе HearMe

Для отображения различных выражений лица в программе HearMe используются три растровых изображения.

1. FaceOpen.bmp (см. рис. 6.12) — изображение лица человека с открытым ртом.
2. FaceClose.bmp (рис. 6.13) — изображение лица человека с закрытым ртом.
3. FaceMid.bmp (рис. 6.14) — изображение лица человека с широко открытым ртом и полуприкрытыми глазами.

Рис. 6.12. Растровое изображение FaceOpen.bmp

Рис. 6.13. Растровое изображение FaceClose.bmp

Рис. 6.14. Растровое изображение FaceMid.bmp

Воспроизведение и анимация

Функция SpeakShow() вызывается каждый раз, когда пользователь нажимает командную кнопку **Слушай** (вариант HEARME_PB) или клавишу [H] (вариант WM_CHAR).

Основным компонентом этой функции является цикл while(1):

```
long lCurrentByte = 0L;
while (1)
      {
```

```
lCurrentByte = sp_PlayF ( lCurrentByte,
                          lCurrentByte + 2000L );

if ( lCurrentByte == 0L )
   {
   break;
   }
    .............................
    .........Анимация...........
    .............................
} /* конец цикла while(1) */
```

В теле цикла while(1) воспроизводится звуковой файл начиная с позиции 0L. Файл воспроизводится фрагментами по 2000 байт. Ниже приведено описание данного цикла.

Функция sp_PlayF() возвращает номер последнего воспроизведенного байта файла. Следовательно, после воспроизведения первых 2000 байт переменная lCurrentByte будет равна 2000L. При следующем воспроизведении будут воспроизведены 2000 байт до позиции 4000. Этот процесс продолжается до тех пор, пока не будет достигнут конец звукового файла.

Если значение второго параметра, передаваемого функции sp_PlayF(), превысит фактический размер файла, то TSEngine воспроизведет файл до конца, а функция sp_PlayF() вернет 0L:

```
if ( lCurrentByte == 0 )
   {
   break; /* прервать цикл while (1) */
   }
```

Для получения эффекта одновременности воспроизведения анимации и звука мы должны быть уверены в том, что время воспроизведения каждого фрагмента звукового файла достаточно мало. Итак, вычислим время воспроизведения каждого фрагмента длиной 2000 байт:

$$\text{Количество секунд} = \frac{\text{Количество байтов} \times 8}{\text{Частота дискретизации}}$$

Для воспроизведения 2000 байт понадобится

$$\text{Количество секунд} = \frac{2000 \times 8}{40000} = 0{,}4 \text{ с} = 400 \text{ мс}$$

Поскольку для процесса вывода требуется менее 1 мс, для человека весь процесс (движение изображений и воспроизведение звука) будет происходить одновременно.

Обратите внимание на то, что операторы, при помощи которых осуществляется анимация, выполняются в реальном масштабе времени в промежутках между воспроизведением фрагментов звукового файла. Поэтому эта часть программы должна быть максимально эффективной. Чем меньше интервал времени между воспроизведением фрагментов, тем лучше.

А теперь рассмотрим фрагмент программы HearMe, в котором осуществляется анимация:

```
if ( iMouthFlag == 0 )
   {
   /*-------------------------
   Рисуем лицо с открытым ртом.
   -------------------------*/
   iMouthFlag = 1;
```

```
    DisplayFaceWithOpen  ( hWnd );
    continue;
    }

if ( iMouthFlag == 1 )
    {
    /*-------------------------
    Рисуем лицо с закрытым ртом.
    -------------------------*/
    iMouthFlag = 2;
    DisplayFaceWithClose ( hWnd );
    continue;
    }

if ( iMouthFlag == 2 )
    {
    /*-----------------------------
    Рисуем лицо с полуоткрытым ртом.
    -----------------------------*/
    iMouthFlag = 0;
    DisplayFaceWithMid ( hWnd );
    continue;
    }
```

Значение флага iMouthFlag отражает текущее состояние выражения лица человека. Значение 0 означает, что рот должен быть открыт, 1 — что рот должен быть закрыт, а 2 — полуоткрыт.

При первой итерации цикла while(1) флаг iMouthFlag установлен в 0. Мы проверяем этот факт оператором if. Затем устанавливаем флаг в 1 и выводим изображение человека с открытым ртом с помощью вызова функции

```
DisplayMouthWithOpen( hWnd );
```

После этого используем оператор continue для возврата управления в начало цикла.

Цикл while(1) выполняется вновь — воспроизводятся следующие 2000 байт. Флаг установлен в 1; следовательно, удовлетворяется условие второго оператора if. Программа устанавливает флаг в 2 и выводит изображение человека с закрытым ртом. Затем вновь используется оператор continue для возврата управления в начало цикла.

Цикл while(1) выполняется снова — воспроизводятся следующие 2000 байт. И поскольку флаг установлен в 2, то удовлетворяется условие третьего оператора if. Программа устанавливает флаг в 0 и выводит изображение человека с полуоткрытым ртом. Затем с помощью оператора continue возвращаем управление в начало цикла.

Процесс повторяется до тех пор, пока функция sp_PlayF() не вернет значение 0L, означающее достижение конца воспроизводимого файла. И тогда мы прерываем выполнение цикла while(1):

```
if ( lCurrentByte == 0L )
    {
    break;
    }
```

Функции DisplayFaceWithOpen(), DisplayFaceWithClose() и DisplayFaceWithMid().

Функции DisplayFaceWithOpen(), DisplayFaceWithClose() и DisplayFaceWithMid() применяются для вывода изображений человека соответственно с открытым, закрытым и полуоткрытым ртом. В каждой из них используется вызов функции BitBlt() для вывода соответствующего растрового изображения.

Обратите внимание на то, что функция SpeakShow() заканчивается вызовом функции DisplayFaceWithClose() и, таким образом, последним на экране появится изображение человека с закрытым ртом.

Модернизация программы HearMe

У вас есть возможность модернизировать программу HearMe. Например, вы можете сгенерировать файлы HearMe, которые будут не только разговаривать вашим голосом, но и выводить ваше лицо. Это реализуется посредством применения относительно недорогих устройств — сканеров, которые позволяют преобразовать изображение на бумаге (так называемую твердую копию) в изображение в формате .BMP. Естественно, вам нужно будет отсканировать несколько своих фотографий: одну, на которой рот закрыт, другую — на которой он открыт и, при желании, еще несколько — с открытыми и закрытыми глазами, с поднятыми бровями и т.п.

Кроме того, можно улучшить программу так, чтобы выражение лица соответствовало произносимым звукам. Например, когда программа воспроизводит звук "I", рот открывается, когда "Up" — рот открывается и резко закрывается и т.п. Для реализации этого эффекта необходимо синхронизировать воспроизведение фонем с анимацией.

Программа Dance

Программа Dance отображает на экране дисплея небольшой спектакль — пару, танцующую под аккомпанемент музыки. В программе используются многие из тех решений, которые мы уже рассмотрели.

Компиляция, компоновка и выполнение программы Dance

Для компиляции и компоновки программы Dance с помощью компилятора фирмы Microsoft необходимо выполнить следующие действия:

- Убедиться в том, что компьютер находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
NMAKE Dance.mak    [Enter]
```

Для компиляции и компоновки программы Dance с помощью компилятора фирмы Borland нужно выполнить такие действия:

- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
MAKE -f Dance.bmk    [Enter]
```

Для запуска программы Dance необходимо:

- Выбрать из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- Использовать диалоговое окно **Пролистать** для выбора файла c:\spSDK\Samp4Win\Dance.exe

Главное окно, которое появляется при запуске приложения, приведено на рис. 6.15. В нем находится изображение пары танцоров. Кроме того, в главном окне расположена командная кнопка **Танцы**.

В главном меню программы Dance (см. рис. 6.16) имеются три команды — **Завершить**, **О программе** и **Инструкции**. Если пользователь выбирает команду **Завершить**, то программа воспроизводит фразу "Good-Bye" и завершает свою работу. Если он выбирает команду

О программе, то выводится диалоговое окно **О программе**. Если пользователь выбирает команду **Инструкции**, то выводится диалоговое окно, которое приведено на рис. 6.17. Если он выполняет щелчок на командной кнопке **Танцы**, пара начинает танцевать.

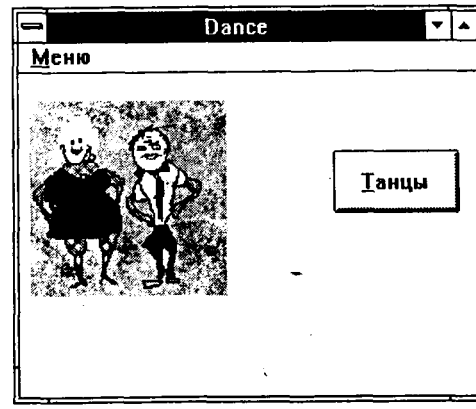


Рис. 6.15. Главное окно программы Dance

Рис. 6.16. Главное меню программы Dance

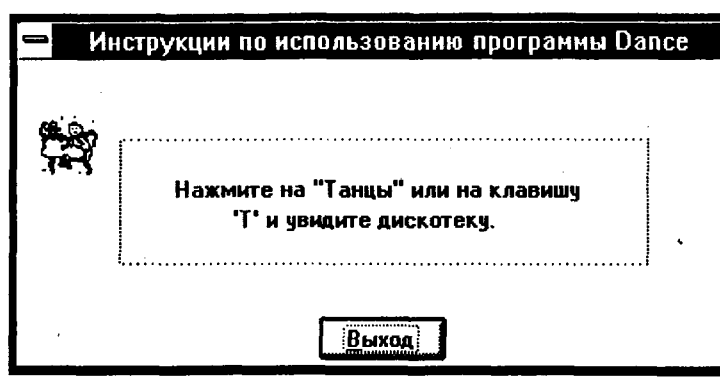


Рис. 6.17. Диалоговое окно команды Инструкции программы Dance

Файлы, входящие в состав программы Dance

Полный комплект программы Dance содержит следующие файлы:

Dance.c	Текст программы на C
Dance.h	Файл .h
Dance.rc	Описание ресурсов программы
Dance.def	Файл определения модуля программы
Dance.mak	Make-файл программы
Dance.ico	Пиктограмма программы
BackGnd.bmp	Растровое изображение, используемое в качестве фона главного окна
Dance0.bmp	Позиция #0 танца
Dance1.bmp	Позиция #1 танца
Dance2.bmp	Позиция #2 танца
Dance3.bmp	Позиция #3 танца
Dance4.bmp	Позиция #4 танца
Dance5.bmp	Заставка

Все эти файлы находятся на прилагаемой дискете и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Ниже приведен полный текст исходных файлов (см. листинги 6.11–6.15).

Листинг 6.11. Dance.h

```
/*================================================
 ИМЯ ФАЙЛА: Dance.h

 (C) Copyright Gurewich 1992, 1993
 ====================  ============================*/

/*--------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc      ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export InstDlgProc  ( HWND, UINT, UINT, LONG ) ;

void DanceShow          ( HWND hWnd );
void DisplayDanceAct0   ( HWND hWnd );
void DisplayDanceAct1   ( HWND hWnd );
void DisplayDanceAct2   ( HWND hWnd );
void DisplayDanceAct3   ( HWND hWnd );
void DisplayDanceAct4   ( HWND hWnd );
void DisplayDanceAct5   ( HWND hWnd );

/*------
 #define
 -------*/
#define IDM_QUIT         1   /* Команда меню Завершить.    */
#define IDM_ABOUT        2   /* Команда меню О программе.  */
#define IDM_INSTRUCTIONS 3   /* Команда меню Инструкции.   */
```

```
#define EXIT_PB   100 /* Кнопка Выход в диалоговом окне. */
#define DANCE_PB 101  /* Кнопка Танцы в главном окне.    */

/*--------------------
 Глобальные переменные
 --------------------*/
char       gszAppName[] = "Dance" ; /* Имя нашего приложения. */
HINSTANCE  ghInst;                  /* Текущий экземпляр.     */
HBRUSH     ghBrushOfBkGnd;          /* Для кисти. */
HBITMAP    ghDanceAct0;             /* Собственно для танцев. */
HBITMAP    ghDanceAct1;             /* Собственно для танцев. */
HBITMAP    ghDanceAct2;             /* Собственно для танцев. */
HBITMAP    ghDanceAct3;             /* Собственно для танцев. */
HBITMAP    ghDanceAct4;             /* Собственно для танцев. */
HBITMAP    ghDanceAct5;             /* Собственно для танцев. */
```

Листинг 6.12. Dance.c

```
/*============================================================
  ПРОГРАММА: Dance.c
  ----------
  (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.
  ОПИСАНИЕ ПРОГРАММЫ:
  -------------------
  Эта программа основана на Generic 1.

  Воспроизведение звукового файла на фоне видео.

 ============================================================*/

/*--------
 #include
 --------*/
/*-------------------------------------------------
 windows.h  требуется для всех приложений Windows.
 -------------------------------------------------*/
#include <windows.h>

/*-------------------------------------------------------------
 sp4Win.h требуется для того, чтобы функции sp_, макросы SP_
 и #define из библиотеки звуковой поддержки TS могли быть
 использованы в этом приложении.
 -------------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-------------------------------------------------------------
 Определения и прототипы, специфические для данного приложения.
 -------------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\Dance.h"

/*-------------------------------------------------------------
 Для функций C, которым требуются стандартные #include файлы C.
 -------------------------------------------------------------*/
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
```

```
/*=================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HWND     hWnd; /* Указатель на окно нашего приложения.          */
MSG      msg;  /* Сообщения, обрабатываемые нашим приложением. */
WNDCLASS wc;   /* Класс окна нашего приложения.                 */

HBITMAP  hBitmapOfBkGnd; /* Указатель на растровое изображение фона. */

/*-------------------------------------------------
 Создаем экземпляр глобальной переменной hInstance.
 -------------------------------------------------*/
ghInst = hInstance;

/*-------------
 Создаем кисть.
 -------------*/
hBitmapOfBkGnd = LoadBitmap ( ghInst, "BackGround"   );
ghBrushOfBkGnd = CreatePatternBrush ( hBitmapOfBkGnd );

/*--------------------------------------
 Инициализируем структуру оконного класса
 и создаем класс.
 --------------------------------------*/
if ( !hPrevInstance )
   {
   /*-----------------------------------
   Если условие выполняется, то это самый
   первый запуск нашего приложения.
   -----------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfDance" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW      ) ;
   wc.hbrBackground = ghBrushOfBkGnd; /* Кисть, которую мы создали с */
                                      /* с помощью растрового изображения.*/
   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = gszAppName ;

  /*-----------------
   Регистрируем окно.
   -----------------*/
   RegisterClass ( &wc );

   }/* конец if(!hPrevInstance) */
```

```
/*------------------------------
 Создаем окно нашего приложения.
 ------------------------------*/
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      50,
                      50,
                      300,
                      250,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );

/*---------------------------------------------
 Отображаем и обновляем окно нашего приложения.
 ---------------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );

/*-------------------------
 Цикл обработки сообщений.
 -------------------------*/
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage  ( &msg );
      }

/*-----------------   ----------------------
 Удаляем фоновый   ъект, который мы создали.
 --------------  -------------------------*/
DeleteObject ( hBitmapOfBkGnd );
DeleteObject ( ghBrushOfBkGnd );

return msg.wParam ;

} /* Конец функции. */
/*======================== конец WinMain() ==================*/


/*=================
 ФУНКЦИЯ: WndProc()
 =================*/
/*-----------------------------
 ОПИСАНИЕ: обработка сообщений.
 -----------------------------*/

long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
```

```
static HDC         hdc;      /* Для вывода. */
static PAINTSTRUCT ps;       /* Для вывода. */
static HDC         hMemDC;   /* Для вывода растрового образа. */

static FARPROC lpfnAboutDlgProc;/* Для диалогового окна О программе.*/
static FARPROC lpfnInstDlgProc ;/* Для диалогового окна Инструкции. */

int    iOpenDanceResult;
int    iOpenHelloResult;

static HWND    hButton;

switch ( message )
       {
       case WM_CREATE:
            /*------------------------
            Начинаем работу со звуком.
            -------------------------*/
            iOpenDanceResult = sp_OpenSession (
                               "c:\\spSDK\\TSfiles\\Music.ts",
                               SP_NON_STAND_ALONE,
                               0L,
                               SP_TS_TYPE);
            /*-----------------------------------------
            Завершаем работу, если не можем открыть файл.
            -----------------------------------------*/
            if ( iOpenDanceResult != SP_NO_ERRORS )
               {
               MessageBox ( NULL,
                            "Невозможно начать работу со звуком.",
                            "Сообщение от dance.exe",
                            MB_ICONINFORMATION );

               /*------------------------
               Завершаем наше приложение.
               -------------------------*/
               SendMessage ( hWnd, WM_DESTROY, 0, 0 );
               }
          /*--------------------------------------
          Получаем lpfnAboutDlgPro, указывающее на
          диалоговое окно О программе.
          --------------------------------------*/
          lpfnAboutDlgProc =
          MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);

          /*--------------------------------------
          Получаем lpfnInstDlgProc, указывающее на
          диалоговое окно Инструкции.
          --------------------------------------*/
          lpfnInstDlgProc =
          MakeProcInstance (( FARPROC) InstDlgProc, ghInst);

          /*-----------------------------------------
          Получаем указатели на растровые изображения
          рисунков, задающих процесс танца.
          -----------------------------------------*/
```

```
        ghDanceAct0  = LoadBitmap ( ghInst, "Dance0"  );
        ghDanceAct1  = LoadBitmap ( ghInst, "Dance1"  );
        ghDanceAct2  = LoadBitmap ( ghInst, "Dance2"  );
        ghDanceAct3  = LoadBitmap ( ghInst, "Dance3"  );
        ghDanceAct4  = LoadBitmap ( ghInst, "Dance4"  );
        ghDanceAct5  = LoadBitmap ( ghInst, "Dance5"  );

        /*-------------------
        Создаем кнопку Танцы.
        ---------------------*/
         hButton = CreateWindow ( "BUTTON",
                                  "&Танцы",
                                  WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
                                  200,
                                  50,
                                  80,
                                  40,
                                  hWnd,
                                  DANCE_PB,
                                  ghInst,
                                  NULL);

        ShowWindow ( hButton, SW_SHOW );

        return 0;
    case WM_PAINT:
        /*-------------------------------------------------------
        Загружаем растровое изображение с базовым изображением.
        -------------------------------------------------------*/
        hdc = BeginPaint ( hWnd, &ps );
        hMemDC = CreateCompatibleDC ( hdc );
        SelectObject ( hMemDC, ghDanceAct0 );
        BitBlt ( hdc,
                 10,
                 20,
                 135,
                 135,
                 hMemDC,
                 0,
                 0,
                 SRCCOPY );
        DeleteDC ( hMemDC );
        EndPaint ( hWnd, &ps );

        return 0;
    case WM_COMMAND:
        /*--------------------
        Обработка команд меню.
        ----------------------*/
        switch (wParam)
               {
               case DANCE_PB:
                    SetFocus  ( hWnd );
                    DanceShow ( hWnd );
                    return 0;
```

```
            case IDM_QUIT:
                 /*-------------------------------------
                 Пользователь выбрал команду Завершить.
                 -------------------------------------*/
                 DestroyWindow (hWnd);
                 return 0L;


            case IDM_ABOUT :
                 /*-------------------------------------
                 Пользователь выбрал команду О программе.
                 -------------------------------------*/
                 DialogBox ( ghInst,
                             "AboutBox",
                             hWnd,
                             lpfnAboutDlgProc );
                 return 0L;


            case IDM_INSTRUCTIONS :
                 /*-------------------------------------
                 Пользователь выбрал команду Инструкции.
                 -------------------------------------*/
                 DialogBox ( ghInst,
                 "InstructionsBox",
                 hWnd,
                 lpfnInstDlgProc );
                 return 0L;
            }/* конец switch (wParam) */

case WM_CHAR:
     /*-----------------------------------
     Пользователь нажал клавишу клавиатуры
     -----------------------------------*/

     /*-----------------------------------
     Преобразовываем ее к верхнему регистру.
     -----------------------------------*/
     wParam = toupper ( wParam );

     switch ( wParam )
            {
            case 210: /* Код русской буквы Т */
                 /*--------------
                 Танец начался!!!
                 ---------------*/
                 DanceShow ( hWnd );
                 return 0;
            }
   return 0L;

case WM_DESTROY:
     /*-------------------------------
     Открываем звуковой файл Hello.ts.
     -------------------------------*/
     iOpenHelloResult =
```

```
            sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                             SP_NON_STAND_ALONE,
                             0L,
                             SP_TS_TYPE ) ;

            if ( iOpenHelloResult == SP_NO_ERRORS )
                {
                /*-----------------------------------
                Воспроизводим часть звукового файла.
                Сколько играть: от 30000 до 40000
                Фраза: "Good-Bye"
                -----------------------------------*/
                sp_PlayF ( 30000L, 40000L ) ;
                }

            /*----------------------------
            Удаляем растровые изображения.
            ----------------------------*/
            DeleteObject ( ghDanceAct0 );
            DeleteObject ( ghDanceAct1 );
            DeleteObject ( ghDanceAct2 );
            DeleteObject ( ghDanceAct3 );
            DeleteObject ( ghDanceAct4 );

            PostQuitMessage (0);
            return 0;
        }/* конец switch (message) */

/*----------------------------
 Сообщение не было обработано.
 ----------------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}/* конец WndProc() */
/*================== конец WndProc() ====================*/

/*=======================
 ФУНКЦИЯ: AboutDlgProc()
 =======================*/

/*-------------------------------------------------
 ОПИСАНИЕ:
 Это функция обработки диалогового окна О программе.
 -------------------------------------------------*/

BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;
```

```
        case WM_COMMAND :
             switch ( wParam )
                    {
                    case IDOK :
                    case IDCANCEL :
                         EndDialog ( hDlg, 0 );
                    return TRUE;
                    }
        }/* конец switch(message) */
return FALSE ;
}/* Конец функции. */
/*================ конец AboutDlgProc() ==============*/

/*======================
 ФУНКЦИЯ: InstDlgProc()
 ======================*/
/*-------------------------------------------------
 ОПИСАНИЕ:
 Это функция, реализующая диалоговое окно Инструкции.
 -------------------------------------------------*/
BOOL FAR PASCAL _export InstDlgProc ( HWND hDlg,
                                      UINT message,
                                      UINT wParam,
                                      LONG lParam )
{
switch ( message )
        {
        case WM_INITDIALOG :
             return TRUE;

        case WM_COMMAND :
             switch ( wParam )
                    {
                    case EXIT_PB :
                    case IDOK :
                    case IDCANCEL :
                         EndDialog ( hDlg, 0 );
                         return TRUE;
                    }
        }
return FALSE ;
}/* Конец функции. */
/*================ конец InstDlgProc() ==============*/


/*========
 ФУНКЦИЯ
 ========*/
/*---------------------
 ОПИСАНИЕ:
 ---------
 Рисует танцующую пару.
 ----------------------*/
void DanceShow ( HWND hWnd )
{
int    iActFlag   = 0;
long   lCurrentByte = 0L;
```

```
DWORD  dwStart;
DWORD  dwFinish;
HWND   hWndCtrl;

/*----------------------
 Запрещаем кнопку Танцы.
 ----------------------*/
hWndCtrl = GetDlgItem ( hWnd, DANCE_PB );
EnableWindow ( hWndCtrl, 0 );

while (1)
      {
      lCurrentByte = sp_PlayF ( lCurrentByte,
                                lCurrentByte + 5000L );

      if ( lCurrentByte == 320000L )
         {
         break;
         }

      if ( iActFlag == 0 )
         {
         /*---------------
         Позиция #1 танца.
         -----------------*/
         iActFlag = 1;
         DisplayDanceAct1 ( hWnd );
         continue;
         }

      if ( iActFlag == 1 )
         {
         /*---------------
         Позиция #2 танца.
         -----------------*/
         iActFlag = 2;
         DisplayDanceAct2 ( hWnd );
         continue;
         }

      if ( iActFlag == 2 )
         {
         /*---------------
         Позиция #3 танца.
         -----------------*/
         iActFlag = 0;
         DisplayDanceAct3 ( hWnd );
         continue;
         }

      }/* конец цикла while(). */

DisplayDanceAct0 ( hWnd );
/*---------------
Крик "Браво-о-о!"
-----------------*/
```

```
sp_PlayF ( 321144L, SP_END_OF_FILE );
DisplayDanceAct4 ( hWnd );

/*---------------
Крик "Браво-о-о!"
-----------------*/
sp_PlayF ( 321144L, SP_END_OF_FILE );
DisplayDanceAct0 ( hWnd );

DisplayDanceAct0 ( hWnd );

/*---------------
Крик "Браво-о-о!"
-----------------*/
sp_PlayF ( 321144L, SP_END_OF_FILE );
DisplayDanceAct5 ( hWnd );

    /*-------
    Задержка.
    ---------*/
    dwStart = GetTickCount ();
    while ( 1 )
          {
          dwFinish = GetTickCount ();
          if ( dwFinish >= dwStart + 4000 )
              break;
          }

DisplayDanceAct0 ( hWnd );

/*----------------------
 Разрешаем кнопку Танцы.
 ----------------------*/
hWndCtrl = GetDlgItem ( hWnd, DANCE_PB );
EnableWindow ( hWndCtrl, 1 );

}/* Конец функции. */
/*============ конец функции ===========*/

/*========
 ФУНКЦИЯ
 ========*/
/*----------------------
 ОПИСАНИЕ:
 ---------
 Вывод позиции #0 танца.
 ----------------------*/
void DisplayDanceAct0 ( HWND hWnd )
{
HDC hdc;
HDC hMemDC;

hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghDanceAct0 );
```

```
BitBlt ( hdc,
         10,
         20,
         135,
         135,
         hMemDC,
         0,
         0,
         SRCCOPY );

DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );
}/* Конец функции. */
/*============ конец функции ===========*/


/*=======
 ФУНКЦИЯ
 ========*/
/*----------------------
 ОПИСАНИЕ:
 ---------
 Вывод позиции #1 танца.
 ----------------------*/
void DisplayDanceAct1 ( HWND hWnd )
{
HDC hdc;
HDC hMemDC;

hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghDanceAct1 );

BitBlt ( hdc,
         10,
         20,
         135,
         135,
         hMemDC,
         0,
         0,
         SRCCOPY );

DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );
}/* Конец функции. */
/*============ конец функции ============*/

/*========
 ФУНКЦИЯ
 ========*/

/*----------------------
 ОПИСАНИЕ:
 ---------
 Вывод позиции #2 танца.
 ----------------------*/
```

```
void DisplayDanceAct2 ( HWND hWnd )
{
HDC hdc;
HDC hMemDC;

hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghDanceAct2 );

BitBlt ( hdc,
         10,
         20,
         135,
         135,
         hMemDC,
         0,
         0,
         SRCCOPY );

DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );
}/* Конец функции. */
/*============ конец функции ===========*/

/*=======
 ФУНКЦИЯ
 ========*/
/*----------------------
 ОПИСАНИЕ:
 ---------
 Вывод позиции #3 танца.
 ----------------------*/
void DisplayDanceAct3 ( HWND hWnd )
{
HDC hdc;
HDC hMemDC;

hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghDanceAct3 );

BitBlt ( hdc,
         10,
         20,
         135,
         135,
         hMemDC,
         0,
         0,
         SRCCOPY );
DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );
}/* Конец функции. */
/*============ конец функции ===========*/
```

```
/*=======
 ФУНКЦИЯ
 ========*/

/*----------------------
 ОПИСАНИЕ:
 ---------
 Вывод позиции #4 танца.
 ----------------------*/
void DisplayDanceAct4 ( HWND hWnd )
{
HDC hdc;
HDC hMemDC;

hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghDanceAct4 );

BitBlt ( hdc,
         10,
         20,
         135,
         135,
         hMemDC,
         0,
         0,
         SRCCOPY );

DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );

}/* Конец функции. */
/*============ конец функции ===========*/


/*=======
 ФУНКЦИЯ
 ========*/
/*----------------------
 ОПИСАНИЕ:
 ---------
 Вывод позиции #5 танца.
 ----------------------*/
void DisplayDanceAct5 ( HWND hWnd )
{
HDC hdc;
HDC hMemDC;

hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghDanceAct5 );
BitBlt ( hdc,
         10,
         20,
         135,
         135,
```

```
          hMemDC,
          0,
          0,
          SRCCOPY );

DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );
}
/*============ конец функции ===========*/
```

Листинг 6.13. Dance.rc

```
/*=============================================
 ИМЯ ФАЙЛА: dance.rc
 ----------
 ОПИСАНИЕ ФАЙЛА:
 ---------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 ============================================*/

/*-------
 #include
 --------*/
#include <windows.h>
#include "Dance.h"

/*---
 Меню
 ----*/
Dance  MENU
BEGIN
   POPUP  "&Меню"
        BEGIN
             MENUITEM "&Завершить", IDM_QUIT
             MENUITEM "&О программе", IDM_ABOUT
             MENUITEM "&Инструкции", IDM_INSTRUCTIONS
        END
END

IconOfDance ICON Dance.ico

/*----------------------
 Растровые изображения.
 ----------------------*/
Dance0         BITMAP Dance0.bmp
Dance1         BITMAP Dance1.bmp
Dance2         BITMAP Dance2.bmp
Dance3         BITMAP Dance3.bmp
Dance4         BITMAP Dance4.bmp
Dance5         BITMAP Dance5.bmp
BackGround     BITMAP BackGnd.bmp
```

```
/*------------------------------------
 Диалоговое окно команды О программе.
 ------------------------------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "О программе Dance"
/* необходим локализованный шрифт MS Sans Serif */
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "&OK", IDOK, 64, 75, 40, 14
    CTEXT           "(C) Copyright Gurewich 1992, 1993",
                    -1, 13, 47, 137, 18
    ICON            "IconOfDance", -1, 14, 12, 18, 20
END

/*------------------------------------
 Диалоговое окно для выдачи инструкций.
 ------------------------------------*/
InstructionsBox DIALOG 41, 45, 222, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Инструкции по использованию программы Dance"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "&Выход", EXIT_PB, 95, 84, 40, 14
    CTEXT           "Нажмите на ""Танцы"" или на клавишу",
                    -1, 37, 38, 152, 13
    CTEXT           "'T' и увидите дискотеку.", -1, 47, 48, 125, 8
    CONTROL         "", -1, "Static", SS_GRAYFRAME, 32, 26, 166, 41
    ICON            "IconOfDance", -1, 7, 18, 18, 20
END
```

Листинг 6.14. Dance.def

```
;======================================
; Файл определения модуля для Dance.c
;======================================

NAME         Dance

DESCRIPTION 'Программа Dance. (C) Copyright Gurewich 1992, 1993'

EXETYPE      WINDOWS

STUB         'WINSTUB.EXE'

CODE  PRELOAD MOVEABLE DISCARDABLE

DATA  PRELOAD MOVEABLE MULTIPLE

HEAPSIZE      1024
STACKSIZE     8192
```

Листинг 6.15. Dance.mak

```
#===========
# Dance.mak
#===========

Dance.exe : Dance.obj Dance.h Dance.def Dance.res
    link /nod Dance.obj, Dance.exe, NUL, \
        slibcew.lib oldnames.lib libw.lib commdlg \
        c:\spSDK\TegoWlib\TegoWin.lib, \
        Dance.def
   rc -t Dance.res

Dance.obj : Dance.c Dance.h
   cl -c -G2sw -Ow -W3 -Zp  Dance.c

Dance.res : Dance.rc Dance.h Dance.ico
   rc -r Dance.rc
```

Функция WndProc() в программе Dance

Ниже рассмотрены различные варианты, которые обрабатываются функцией WndProc().

Открытие сеанса работы со звуком

В варианте WM_CREATE мы открываем сеанс работы со звуковым файлом типа TS Music.ts, находящимся в директории c:\spSDK\Samp4Win\, чтобы создать неавтономное приложение:

```
   case WM_CREATE:
       /*-------------------------
       Начинаем работу со звуком.
       -------------------------*/
       iOpenDanceResult = sp_OpenSession (
                          "c:\\spSDK\\TSfiles\\Music.ts",
                          SP_NON_STAND_ALONE,
                          0L,
                          SP_TS_TYPE);
       /*-------------------------------------------
       Завершаем работу, если не можем открыть файл.
       -------------------------------------------*/
       if ( iOpenDanceResult != SP_NO_ERRORS )
           {
           MessageBox ( NULL,
                        "Невозможно начать работу со звуком.",
                        "Сообщение от dance.exe",
                         MB_ICONINFORMATION );

           /*-------------------------
           Завершаем наше приложение.
           -------------------------*/
           SendMessage ( hWnd, WM_DESTROY, 0, 0 );
           }
   ..............................
   ..............................
   ..............................
   return 0;
```

Отображение танца

Фрагмент программы, в котором осуществляется отображение танца, реализован в виде функции DanceShow(). WndProc() вызывает эту функцию каждый раз, когда пользователь нажимает командную кнопку **Танцы**.

Вариант DANCE_PB

В варианте DANCE_PB обрабатываем щелчок на командной кнопке **Танцы**. При этом вызывается функция DanceShow():

```
case DANCE_PB:
     SetFocus ( hWnd );
     DanceShow ( hWnd );
     return 0;
```

Вариант WM_CHAR

В варианте WM_CHAR обрабатываем нажатие пользователем клавиши [T] и также вызываем функцию DanceShow():

```
case WM_CHAR:
     /*-------------------------------
     Преобразование в верхний регистр.
     -------------------------------*/
     wParam = toupper ( wParam );

     switch ( wParam )
            {
            case 210:  /* Код русской буквы T */
                 /*--------------
                 Танец начался!!!
                 ---------------*/
                 DanceShow ( hWnd );
                 return 0;
            }
     return 0L;
```

Функция DanceShow()

Функция DanceShow() отображает танец следующим образом:

```
void DanceShow ( hWnd )
{
while (1)
      {
       ........................
       .... Отображение танца ....
       ........................
      }
.....................................
......Последняя картинка танцев.....
.....................................
}
```

Цикл while(1) в функции DanceShow()

В функции DanceShow() используется звуковой файл Music.ts. Этот файл является результатом слияния двух файлов, которое было осуществлено с помощью TS Sound Editor. А так как мы предполагаем, что у вас этого редактора нет, то приводим структуру данного файла:

```
========================        ====================
Байтовые координаты             Звуковой фрагмент
========================        ====================
0 - 321144                      Музыка
321144 - SP_END_OF_FILE         Аплодисменты и крики "Браво!!!"
```

В цикле while(1) музыка воспроизводится фрагментами по 5000 байт каждый:

```
lCurrentByte = sp_PlayF ( lCurrentByte,
                          lCurrentByte + 5000L );
```

Файл воспроизводится до позиции 320000, после чего происходит выход из цикла.

```
if ( lCurrentByte == 320000L )
   {
   break;
   }
```

Размер файла — 321144 байта, но последние 1144 байта не "прозвучат", так как воспроизведение файла осуществляется фрагментами по 5000 байт каждый.

Отображение танца производится с помощью трех фреймов: Act1, Act2 и Act3. Эти фреймы хранятся в растровых файлах Dance1.bmp, Dance2.bmp и Dance3.bmp (см. рис. 6.18).

Рис. 6.18. Растровые файлы программы Dance

У нас есть флаг (который называется iActFlag), показывающий, какой именно фрейм отображен в данный момент. Если флаг установлен в 0, то отображен фрейм Act1. Если он установлен в 1, то — Act2. А если iActFlag равен 2, то отображаем Act3. Флаг изменяется после воспроизведения каждого фрагмента звукового файла размером в 5000 байт. Мы реализовали этот процесс с помощью четырех операторов if:

```
if ( iActFlag == 0 )
   {
   /*---------------
   Позиция #1 танца.
   -----------------*/
   iActFlag = 1;
   DisplayDanceAct1 ( hWnd );
   continue;
   }
```

```
if ( iActFlag == 1 )
   {
   /*---------------
   Позиция #2 танца.
   -----------------*/
   iActFlag = 2;
   DisplayDanceAct2 ( hWnd );
   continue;
   }

if ( iActFlag == 2 )
   {
   /*---------------
   Позиция #3 танца.
   -----------------*/
   iActFlag = 0;
   DisplayDanceAct3 ( hWnd );
   continue;
   }
```

Перед прерыванием цикла while(1) вызовом функции DisplayDanceAct0 () выводим изображение пары, которая не танцует:

```
DisplayDanceAct0 ( hWnd );
```

Функция DisplayDanceAct0 () выводит растровый файл Dance0.bmp (см. рис. 6.19).

Рис. 6.19. Графический файл Dance0.bmp

Затем мы предоставляем аудитории возможность покричать "Браво!!!", воспроизводя соответствующую часть звукового файла:

```
/*---------------
Крик "Браво-о-о!"
-----------------*/
sp_PlayF ( 321144L, SP_END_OF_FILE );
```

И действительно, танцоры заслужили благодарность зрителей. Благодарный поклон танцоров реализуем посредством вызова функции DisplayDanceAct4 ():

```
DisplayDanceAct4 (hWnd);
```

Функция DisplayDanceAct4 () выводит растровый файл Dance4.bmp (см. рис. 6.20).

После этого еще раз воспроизводим крики "Браво!!!" и вызовом функции Display-DanceAct0 () заставляем танцоров занять исходную позицию.

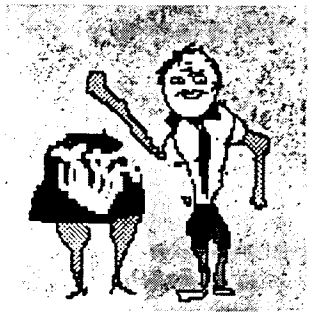


Рис. 6.20. Растровый файл Dance4.bmp

В конце выводим кадр, где указывается автор анимации (как в настоящем кино). Это делается вызовом функции DisplayDanceAct5 ():

```
DisplayDanceAct5 (hWnd);
```

Функция DisplayDanceAct5 () выводит растровый файл Dance5.bmp (см. рис. 6.21).

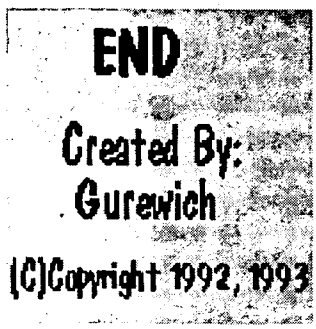


Рис. 6.21. Растровый файл Dance5.bmp

Последний кадр задерживаем на экране на четыре секунды, реализуя эту задержку следующим образом:

```
/*----------
 Задержка.
 ---------*/
 dwStart = GetTickCount ();
 while ( 1 )
       {
       dwFinish = GetTickCount ();
       if ( dwFinish >= dwStart + 4000 )
            break;
       }
```

Представление заканчивается вызовом функции DisplayDanceAct0 (), которая переводит танцоров в исходную позицию — они снова готовы танцевать.

Глава 7. Синхронизация вывода текста и воспроизведения речи

В этой главе рассматриваются возможности одновременного вывода текста и воспроизведения речи.

Программа Press

Главное окно программы Press содержит командную кнопку **Нажмите кнопку или 'P'**. Если пользователь нажимает эту кнопку, программа воспроизводит фразу "Press any key to continue" и одновременно выводит идентичный текст. Вывод текста синхронизирован с воспроизведением речи (т.е. когда произносится "Press", выводится слово "Press", при воспроизведении "any" — выводится "any" и т.д.).

Мы рекомендуем перед рассмотрением текста программы откомпилировать и запустить ее — это поможет вам лучше понять работу программы.

Компиляция, компоновка и запуск программы Press

Для компиляции и компоновки программы Press с помощью компилятора фирмы Microsoft необходимо выполнить следующие действия:

- Убедиться в том, что компьютер находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
NMAKE Press.mak    [Enter]
```

Для компиляции и компоновки программы Press с помощью компилятора фирмы Borland нужно выполнить такие действия:

- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
MAKE -f Press.bmk    [Enter]
```

Для запуска программы Press необходимо:

- Выбрать из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- Использовать диалоговое окно **Пролистать** для выбора файла

```
c:\spSDK\Samp4Win\Press.exe
```

Появляющееся при запуске главное окно приложения приведено на рис. 7.1. Как вы можете видеть, оно содержит командную кнопку **Нажмите кнопку или 'P'**.

Главное меню программы Press (см. рис. 7.2) содержит две команды — **Завершить** и **О программе**. Если пользователь выбирает команду **О программе**, то выводится диалоговое окно **О программе Press**, которое приведено на рис. 7.3. Если он выбирает команду **Завершить**, то программа произносит фразу "Good-Bye" и завершает свою работу.

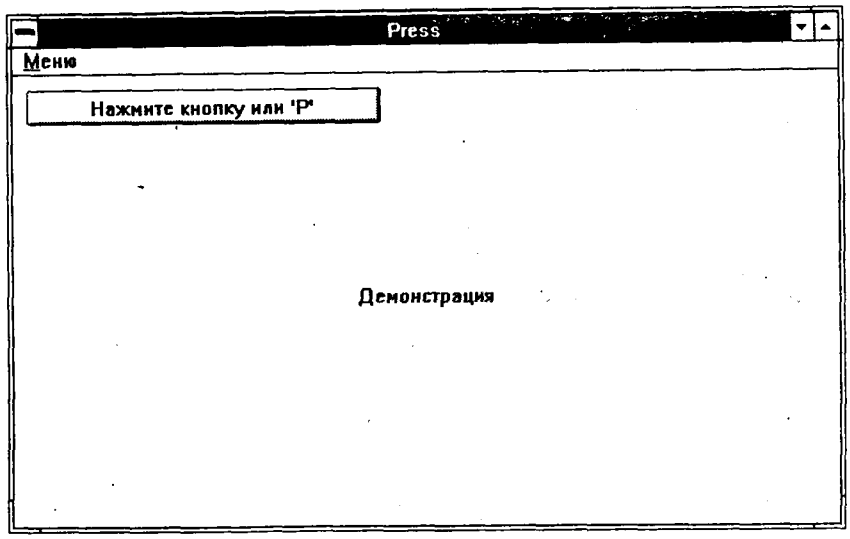


Рис. 7.1. Главное окно программы Press

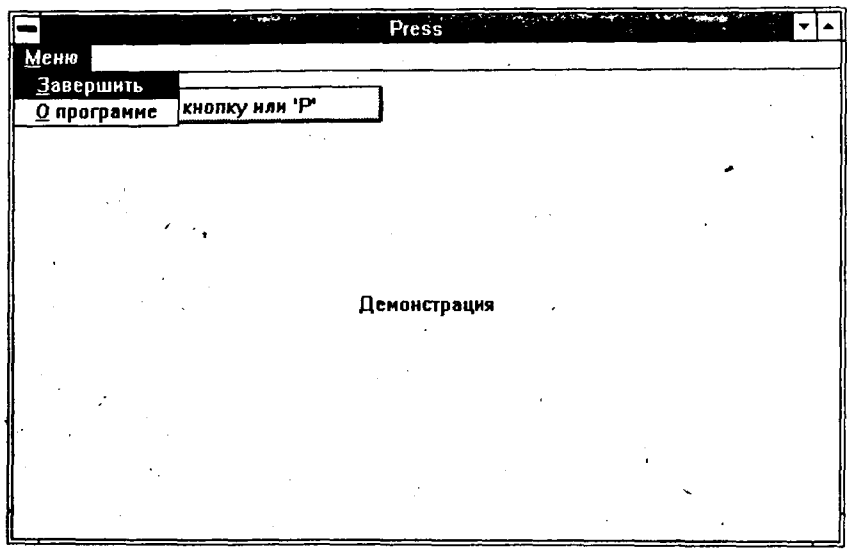


Рис. 7.2. Главное меню программы Press

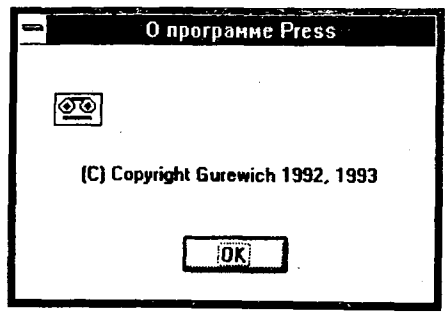


Рис. 7.3. Диалоговое окно О программе Press

Если пользователь нажимает командную кнопку, находящуюся в главном окне (или клавишу [P] на клавиатуре компьютера), программа воспроизводит фразу "Press any key to continue". Синхронно с воспроизведением каждое слово фразы отображается в главном окне.

Файлы, входящие в состав программы Press

Полный комплект программы Press содержит следующие файлы:

Press.h	Файл .h
Press.c	Текст программы на C
Press.rc	Описание ресурсов программы
Press.def	Файл определения модуля программы
Press.mak	Make-файл программы
Tape.ico	Пиктограмма программы

Все эти файлы находятся на прилагаемой к книге дискете и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Функция WndProc() программы Press.c

Функция WndProc() программы Press.c обрабатывает события, которые возникают в процессе выполнения программы.

Вариант WM_CREATE

В варианте WM_CREATE мы открываем сеанс работы со звуковым файлом типа S C:\spSDK\SFiles\Press.s, чтобы создать неавтономное приложение:

```
iOpenResult = sp_OpenSession (
              "c:\\spSDK\\Sfiles\\Press.s",
              SP_NON_STAND_ALONE,
              0L,
              SP_S_TYPE);
```

Если успешного открытия сеанса не произошло, то завершаем приложение:

```
if ( iOpenResult != SP_NO_ERRORS )
   {
   MessageBox ( NULL,
                "Невозможно начать работу со звуком.",
                "Сообщение от press.exe",
                MB_ICONINFORMATION );

    SendMessage ( hWnd,
                  WM_DESTROY,
                  0,
                  0 );
   }
```

Получаем lpfn-указатель на диалоговое окно **О программе Press**, а затем создаем и выводим командную кнопку, располагающуюся в главном окне:

```
hButton =
CreateWindow ( "BUTTON",
               "Нажмите кнопку или 'P'",
```

```
                WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                10,
                10,
                250,
                25,
                hWnd,
                INSTRUCTION_PB,
                ghInst,
                NULL);

    ShowWindow ( hButton, SW_SHOW );
```

Девятый параметр, передаваемый функции CreateWindow(), — INSTRUCTION_PB — определен в файле Press.h.

Вариант WM_CHAR

Сообщение WM_CHAR поступает всякий раз, когда пользователь нажимает клавишу на клавиатуре компьютера. При поступлении этого сообщения мы преобразовываем полученный символ в соответствующий символ верхнего регистра и определяем, какая клавиша была нажата:

```
switch ( wParam )
       {
       case 'P':
            /*-----------------------
            Воспроизводим инструкции.
            -----------------------*/
            PlayInstruction ( hWnd );
            return 0L;
       }
```

Если пользователь нажал клавишу [P], то вызываем функцию PlayInstruction(). Аналогично, если он нажимает командную кнопку INSTRUCTION_PB, то мы определяем это в варианте INSTRUCTION_PB и также вызываем функцию PlayInstruction():

```
case INSTRUCTION_PB:
     /*-----------------------------------
     Пользователь выбрал команду Инструкции.
     -----------------------------------*/
     PlayInstruction ( hWnd );

     return 0L;
```

Функция PlayInstruction()

Прототип функции PlayInstruction() определен в файле Press.h:

```
void PlayInstruction (hWnd);
```

Мы передаем этой функции hWnd, поскольку она осуществляет вывод в главное окно текстовых сообщений. Следовательно, функции нужно передавать значение hWnd главного окна.

В функции PlayInstruction() сначала инициализируем пробелами строковую переменную sSpaces[]. Впоследствии этот массив будет использоваться для стирания строки текста:

```
/*-----------------------
 Создаем строку пробелов.
 -----------------------*/
 for ( i=0; i<100; i++ )
     sSpaces[i] = ' ';
     sSpaces[i] = '\0';
```

Для вывода текста в функции PlayInstruction () используется функция TextOut(). Значит, мы должны получить переменные, которые нужно передавать TextOut() в качестве параметров:
Для получения hDC используем функцию GetDC():

```
hDC = GetDC(hWnd):
```

Обновляем структуры tm (в структуре tm находятся размеры текущего шрифта текста):

```
GetTextMetrics ( hDc, &tm );
```

Горизонтальный размер шрифта находится в элементе tmAveCharWidth структуры tm, а вертикальные размеры — в элементах tmWeight и tmExternalLeading той же структуры:

```
cxChar = tm.tmAveCharWidth ;
cyChar = tmWeight + tmExternalLeading;
```

В программе Press не используется вертикальный размер шрифта. Практически cyChar используется только тогда, когда текст выводится сразу в нескольких строках.
Значение cxChar используется позже, при вызове функции TextOut().

А теперь, когда получены значения всех переменных, необходимых для вызова функции TextOut(), можно вывести слово "Press":

```
TextOut ( hDC,
          50,
          50,
          "Press ",
          6 );
```

и воспроизвести слово "Press":

```
sp_PlayF ( 0L, 3896L );
```

Как мы узнали, что фрагмент, отвечающий за слово "Press", находится в позициях 0–3896 файла C:\spSDK\SFiles\Press.s? Для этого использовался TS Editor. Но поскольку мы предполагаем, что у вас еще нет редактора TS Editor, то приводим расположение фрагментов в файле Press.s:

Байтовые координаты	*Фраза*
0 – 3896	Press
3896 – 6224	any
6224 – 10137	key
10137 – 11228	to
11228 – SP_END_OF_FILES	continue

Вы, возможно, обратили внимание на то, что позиция 3896 используется в качестве конечной для слова "Press" и одновременно в качестве начальной для слова "any". Фактически начальной позицией "any" является 3897, но мы пренебрегаем одним звуковым байтом. Длительность воспроизведения одного байта файла типа S можно определить по формуле:

$$\text{Длительность в секундах для типа S} = \frac{\text{Количество байтов} \times 8}{\text{Частота дискретизации}}$$

Файл Press.s был записан с частотой дискретизации 40000 Гц. Используя указанную формулу, получим длительность воспроизведения одного байта файла Press.s:

$$\text{Длительность} = \frac{1 \times 8}{40000} = 0{,}0002 \text{ с} = 0{,}2 \text{ мс}$$

Как видите, одним байтом данного файла можно смело пренебречь.

В остальной части функции PlayInstruction() содержатся аналогичные фрагменты для вывода и воспроизведения слов any, key, to и continue.

В конце удаляем весь выведенный текст посредством отображения строки, содержащей пробелы:

```
TextOut ( hDC,
          50,
          50,
          sSpaces,
          lstrlen( sSpaces) );
```

Вариант WM_DESTROY

Перед завершением программы воспроизводим фразу "Good-Bye":

```
case WM_DESTROY:
     iOpenResult =
     sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                      SP_NON_STAND_ALONE,
                      0L,
                      SP_TS_TYPE);

     if ( iOpenResult == SP_NO_ERRORS )
        {
        /*-----------------------------------
        Воспроизводим часть звукового файла.
        Сколько играть: от 30000 до 40000
        Фраза: "Good-Bye"
        -----------------------------------*/
        sp_PlayF ( 30000L, 40000L );
        }
     PostQuitMessage (0);
     return 0;
```

Программы, аналогичные Press

Техника, продемонстрированная в программе Press, может быть применена в любом приложении, использующем аудиовизуальные средства помощи. Например, ваша программа может отображать диалоговое окно с командной кнопкой, вызывающей соответствующие аудиовизуальные средства помощи.

В некоторых программах, возможно, понадобится производить отображение и воспроизведение текста аудиовизуальной помощи без нажатия командной кнопки. Например, вы создаете программу, которая в процессе своей работы осуществляет вывод на печать. Обнаружив, что в принтер не заправлена бумага, она отображает на экране и воспроизводит следующий текст: "Нет бумаги! Пожалуйста, заправьте в принтер бумагу".

Программа PlzWait

Рассмотрим еще одну программу, выводящую изображение синхронно с воспроизведением звука, — программу PlzWait. В отличие от программы Press программа PlzWait выводит и перемещает текст с помощью растровых изображений.

После того как пользователь выполнит щелчок на командной кнопке **Слушай**, программа PlzWait выводит и воспроизводит фразу "Please wait" ("Подождите, пожалуйста").

Компиляция и компоновка программы PlzWait

Для компиляции и компоновки программы PlzWait с помощью компилятора фирмы Microsoft необходимо выполнить следующие действия:

- Убедиться в том, что компьютер находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
NMAKE PlzWait.mak   [Enter]
```

Для компиляции и компоновки программы PlzWait с помощью компилятора фирмы Borland нужно выполнить такие действия:

- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
MAKE -f PlzWait.bmk   [Enter]
```

Запуск программы PlzWait

Для запуска программы PlzWait необходимо:

- Выбрать из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- Использовать диалоговое окно **Пролистать** для выбора файла

```
c:\spSDK\Samp4Win\PlzWait.exe
```

Главное окно приложения, которое выводится на экран при запуске, приведено на рис. 7.4. Меню программы PlzWait (см. рис. 7.5) содержит три команды — **Завершить**, **О программе** и **Инструкции.**

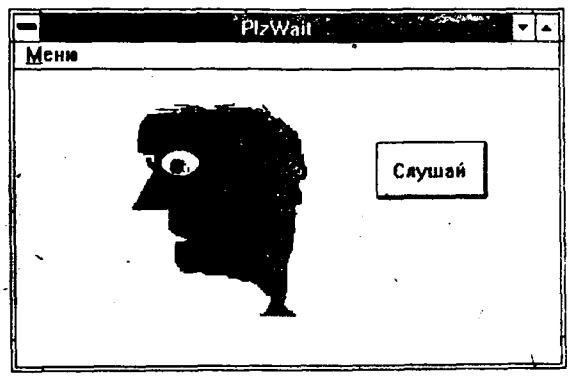


Рис. 7.4. Главное окно программы PlzWait

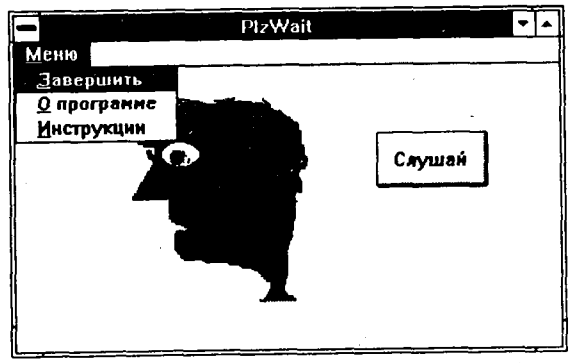


Рис 7.5. Меню программы PlzWait

Если пользователь выбирает команду **О программе**, то появляется диалоговое окно **О программе PlzWait**, приведенное на рис. 7.6. Если он выбирает команду **Инструкции**, то увидит на экране диалоговое окно **Инструкции по использованию программы PlzWait** (см. рис. 7.7).

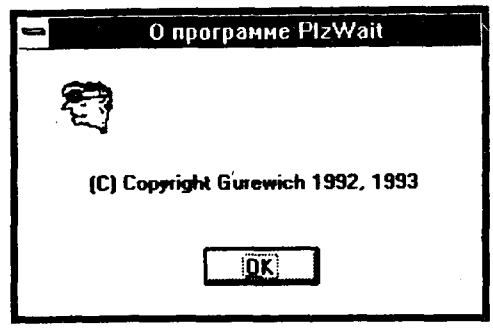


Рис. 7.6. Диалоговое окно О программе PlzWait

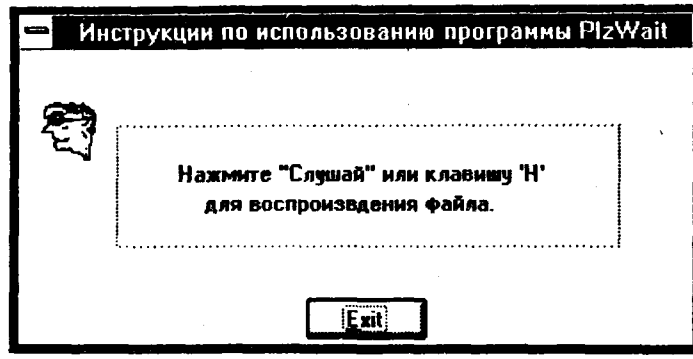


Рис 7.7. Диалоговое окно Инструкции по использованию программы PlzWait

Главное окно программы PlzWait содержит командную кнопку **Слушай**. Когда пользователь нажимает эту кнопку, человек, изображенный в главном окне, открывает рот и в окне появляется слово "Please", которое затем воспроизводится; потом появляется и воспроизводится слово "Wait" (см. рис. 7.8). При этом человек открывает и закрывает рот. Этот процесс повторяется дважды (т.е. фраза "Please Wait" воспроизводится два раза).

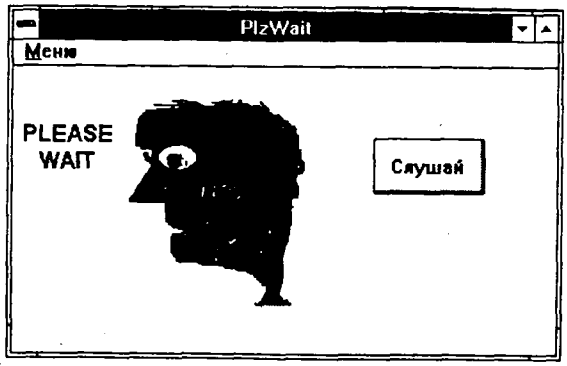


Рис. 7.8. Вывод растровых изображений программой PlzWait

Эта реплика может быть вставлена в ваши программы для воспроизведения во время наиболее длительных операций: надо просто исключить ожидание нажатия командной кнопки **Слушай** и сразу же воспроизводить указанную фразу во время выполнения наиболее длительных операций.

Файлы, входящие в состав программы PlzWait

Полный комплект программы PlzWait содержит следующие файлы:

PlzWait.h	Файл .h
PlzWait.c	Текст программы на C
PlzWait.rc	Описание ресурсов программы
PlzWait.def	Файл определения модуля программы
PlzWait.mak	Make-файл программы
PlzWait.ico	Пиктограмма программы
Wait1.bmp	Изображение человека
Wait2.bmp	Изображение человека с закрытым ртом
Wait3.bmp	Изображение человека с открытым ртом
Wait4.bmp	Изображение слова "Please"
Wait5.bmp	Изображение слова "Wait"
BackGnd.bmp	Растровое изображение, которое используется в качестве фона

Все эти файлы находятся на прилагаемой дискете и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Функция WndProc() программы PlzWait

Функция WndProc() используется для обработки событий, которые возникают в процессе выполнения программы PlzWait.

Вариант WM_CREATE

В варианте WM_CREATE мы открываем сеанс работы со звуковым файлом типа S C:\spSDK\SFiles\Wait.s, чтобы создать неавтономное приложение:

```
case WM_CREATE:
     iOpenResult = sp_OpenSession (
                   "c:\\spSDK\\Sfiles\\Wait.s",
                   SP_NON_STAND_ALONE,
                   0L,
                   SP_S_TYPE);
```

Если успешного открытия сеанса не произошло, то завершаем приложение:

```
if ( iOpenResult != SP_NO_ERRORS )
   {
   MessageBox ( NULL,
                "Невозможно начать работу со звуком.",
                "Сообщение от PlzWait.exe",
                MB_ICONINFORMATION );

   SendMessage ( hWnd,
                 WM_DESTROY,
                 0,
                 0 );
    }
```

С помощью вызова функции MakeProcInstance() получаем lpfn-указатели на диалоговые окна **О программе** и **Инструкции**, а затем — указатели на все растровые изображения, которые используются в программе:

```
ghFullFace      = LoadBitmap ( ghInst, "FullFace" );
ghMouthClosed   = LoadBitmap ( ghInst, "MouthClosed" );
ghMouthOpen     = LoadBitmap ( ghInst, "MouthOpen" );
ghPlease        = LoadBitmap ( ghInst, "Please" );
ghWait          = LoadBitmap ( ghInst, "Wait" );
```

Соответствие растровых изображений файлам задано в файле ресурсов PlzWait.rc:

```
FullFace       BITMAP Wait1.bmp
MouthClosed    BITMAP Wait2.bmp
MouthOpen      BITMAP Wait3.bmp
Please         BITMAP Wait4.bmp
Wait           BITMAP Wait5.bmp
BackGround     BITMAP BackGnd.bmp
```

И последнее, что мы делаем в варианте WM_CREATE — это создаем командную кнопку **Слушай**.

Вариант WM_PAINT

В варианте WM_PAINT мы отображаем в главном окне приложения файл Wait1.bmp (см. рис. 7.9):

```
case WM_PAINT:
     /*--------------------------
     Рисуем лицо с закрытым ртом.
     --------------------------*/
     hdc = BeginPaint ( hWnd, &ps );
     hMemDC = CreateCompatibleDC ( hdc );
     SelectObject ( hMemDC, ghFullFace );
     BitBlt ( hdc,
              10,
              20,
              200,
              150,
```

```
            hMemDC,
            0,
            0,
            SRCCOPY );
    DeleteDC ( hMemDC );
    EndPaint ( hWnd, &ps );

    return 0;
```

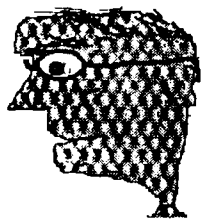


Рис. 7.9. Файл Wait1.bmp

Варианты HEARME_PB и WM_CHAR

После того как пользователь выполнил щелчок на командной кнопке **Слушай**, генерируется сообщение WM_COMMAND. При этом wParam устанавливается в HEARME_PB. В этом варианте вызовом функции SetFocus() мы возвращаем фокус в главное окно и вызываем функцию SpeakShow():

```
case HEARME_PB:
     SetFocus   ( hWnd );
     SpeakShow  ( hWnd );
     return 0;
```

Аналогично, если пользователь нажимает на клавиатуре компьютера клавишу [H], то мы преобразовываем полученный символ в соответствующий символ верхнего регистра и вызываем функцию SpeakShow():

```
case WM_CHAR:
     /*---------------------------------
     Преобразование к верхнему регистру.
     ---------------------------------*/
     wParam = toupper ( wParam );

     switch ( wParam )
            {
            case 'H':
                 /*-----------------------
                 Человек начинает говорить.
                 -----------------------*/
                 SpeakShow ( hWnd );
                 return 0;
            }

     return 0L;
```

Функция SpeakShow()

Основная часть программы — функция SpeakShow() — является тем полем деятельности, где вы можете применить все свое воображение и творческую активность.

Прототип функции SpeakShow() описан в файле PlzWait.h:

```
void SpeakShow ( HWND hWnd );
```

Функции SpeakShow() нужно передавать hWnd, так как она выводит изображение в главное окно и, следовательно, ей требуется указатель на это окно.

Поскольку мы повторяем процесс дважды, то основной код функции помещается в теле цикла for():

```
for ( iCounter = 0; iCounter < 2; iCounter++ )
    {
    ....................................
    ..........Текст шоу..................
    ....................................
    DisplayFace ( hWnd );
    }
```

Функция DisplayFace() вызывается после завершения каждой итерации процесса. Эта функция выводит файл Wait1.bmp (см. рис. 7.9). Как видите, рот человека закрыт, как и должно быть в момент завершения воспроизведения фразы.

Организация процесса демонстрации

Демонстрация начинается выводом изображения рта человека в открытом состоянии. Мы делаем это вызовом функции DislpayOpenMouth():

```
/*-------------------------------
 Рисуем человека с открытым ртом.
-------------------------------*/
DislpayOpenMouth ( hWnd );
```

С целью экономии дискового пространства функция DislpayOpenMouth() выводит очень маленькое изображение — Wait3.bmp (см. рис. 7.10) — только открытого рта.

Воспроизводим фразу "Please":

```
/*---------------------------
 Воспроизводим фразу "Please".
-----------------------------*/
sp_PlayF ( SP_START_OF_FILE, 4700L );
```

Затем вызовом функции DisplayPlease() выводим растровый файл Wait4.bmp:

```
/*----------------------
 Выводим слово "Please".
------------------------*/
DisplayPlease (hWnd);
```

В растровом файле Wait4.bmp (см. рис. 7.10) находится текст "Please".

После вывода фразы "Please" вызовом функции DisplayCloseMouth() мы "заставляем" изображенного на экране человека закрыть рот:

```
/*-----------------------------------------
 Выводим изображение человека с закрытым ртом.
-------------------------------------------*/
DisplayClosedMouth ( hWnd );
```

Функция DisplayClosedMouth() вызывает функцию BitBlt() для вывода растрового файла Wait2.bmp. В этом файле находится небольшое изображение — закрытого рта (см. рис. 7.10).

Задержку на полсекунды (500 мс) реализуем с помощью цикла

```
/*--------
 Задержка.
 ---------*/
 dwStart = GetTickCount ();

 while ( 1 )
       {
       dwFinish = GetTickCount ();
       if ( dwFinish = dwStart + 500 )
            break;
       }
```

Теперь уже все готово для отображения и воспроизведения слова "Wait". Начинаем с открытия рта человека вызовом функции DisplayOpenMouth():

```
/*---------------------
 Человек открывает рот.
 ----------------------*/
 DisplayOpenMouth ( hWnd);
```

а затем воспроизводим слово "Wait":

```
/*-----------------
 Произносит "Wait".
 ------------------*/
 sp_PlayF ( 4700L, SP_END_OF_FILE );
```

И наконец, вызываем функцию DisplayWait() для отображения текста "Wait":

```
/*----------------
 Отображаем "Wait"
 -----------------*/
 DisplayWait ( hWnd );
```

Wait5.bmp — это растровое изображение текста "Wait" (см. рис. 7.10).

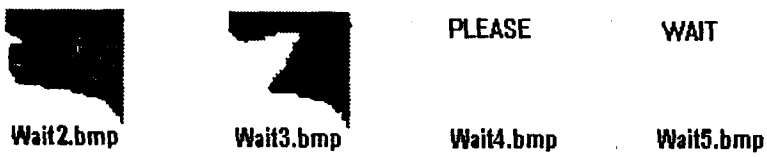


Рис. 7.10. Растровые изображения, используемые в программе PlzWait

Перед возвратом в начало цикла "заставляем" человека закрыть рот:

```
/*---------------------
 Человек закрывает рот.
 ----------------------*/
 DisplayClosedMouth ( hWnd );
```

и организовываем небольшую задержку в 500 мс:

```
/*--------
 Задержка.
 ---------*/
 dwStart = GetTickCount ();
```

```
while ( 1 )
      {
      dwFinish = GetTickCount ();
      if ( dwFinish = dwStart + 500 )
           break;
      }
```

Функция DisplayPlease()

Функция DisplayPlease() служит для вывода слова "Please". Начинается она с подготовки переменных, которые должны быть переданы функции SelectObject(), и вызова SelectObject():

```
hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, ghPlease );
```

Объект, выбранный функцией SelectObject(), — это растровое изображение текста "Please", что и указано во втором параметре SelectObject().

При помощи функции DisplayPlease() реализовано следующее: слово "Please" как будто вылетает изо рта человека (перемещается влево), а затем вертикально поднимается вверх. Мы организуем эти действия посредством двух циклов for(). Первый из них — для перемещения слова "Please" в горизонтальном направлении:

```
iYStep = 110;
for ( iXStep = 50;
   iXStep >= 10;
   iXStep-- )
    {
    BitBlt ( hdc,
             iXStep,
             iYStep,
             60,
             40,
             hMemDC,
             0,
             0,
             SRCCOPY );

   }/* конец цикла for(). */
```

Начальные координаты изображения Wait4.bmp (текст "Please") в пикселах — $x = 50$, $y = 110$. В цикле for () реализуется горизонтальное перемещение текста посредством уменьшения координаты x с последующим выводом растрового образа. Как вы можете видеть из рис. 7.11, пиксельные координаты (50, 110) соответствуют позиции, которая находится слева перед ртом человека.

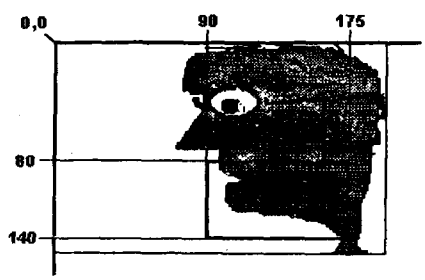


Рис. 7.11. Пиксельные координаты области рта

Во втором цикле for () осуществляется перемещение растрового изображения текста "Please" (Wait4.bmp) в вертикальном направлении. Мы делаем это так: изображение выводится в цикле функцией BitBlt(), а координата *y* уменьшается при каждой итерации:

```
iYStep = 110;
for ( iXStep = 50;
    iXStep >= 10;
    iXStep-- )
    {
    BitBlt ( hdc,
             iXStep,
             iYStep,
             60,
             40,
             hMemDC,
             0,
             0,
             SRCCOPY );

    }/* конец цикла for(). */
```

При каждой итерации слово "Please" перемещается на новую позицию (на одну позицию влево или на одну позицию вверх). В процессе перемещения слова "Please" нет необходимости каждый раз восстанавливать исходный фон того участка экрана, который занимало изображение слова после предыдущей итерации. Мы можем себе это позволить, поскольку фон изображения Wait4.bmp — желтый, т.е. совпадает с фоном главного изображения (мы специально создали файл Wait4.bmp с таким фоном). При перемещении слова "Please" желтый край этого изображения накладывается на часть предыдущего изображения "Please" и автоматически стирает его. Таким образом, нам не потребовались специальные операторы для сохранения и последующего восстановления исходного изображения фона.

Другие способы реализации демонстрации

Вы можете поэкспериментировать с функцией SpeakShow(), изменяя последовательность событий. Например, можно сделать так, чтобы человек сначала произносил текст (то есть воспроизводился звук) и только после этого данный текст возникал на экране. Порядок событий жестко не регламентирован — все зависит от вашего вкуса и желания.

Применение программы Paintbrush

Все изображения, которые использует программа PlzWait, были созданы при помощи графического редактора Paintbrush, входящего в комплект поставки MS Windows. Поскольку пиксельные координаты растровых изображений очень важны, мы выбрали в меню **Просмотр** редактора Paintbrush команду **Координаты курсора**. После этого пиксельные координаты перемещающегося курсора стали отображаться в правом верхнем углу главного окна Paintbrush. При вырезании, копировании и выполнении других манипуляций использовались клавиши со стрелками для установки курсора в точно заданную позицию.

При сохранении изображения мы убеждались в том, что оно сохраняется именно в формате .BMP. Мы также ограничили размер графических файлов заданием высоты и ширины каждого изображения из .BMP-файла. Это было сделано при помощи команды **Атрибуты образа** меню **Параметры** и заданием в диалоговом окне высоты и ширины изображения в файле .BMP.

Основные рекомендации по созданию демонстрации

Если вы будете создавать собственную демонстрацию, то рекомендуем начать с разработки функций высокого уровня, таких как DisplayCloseMouth(), DispleyOpenMouth() и т.п. Когда функции высокого уровня будут созданы и отлажены, вы сможете использовать их в своей программе, что сделает ее более читабельной и легкой для понимания. Кроме того, вам будет проще экспериментировать и, таким образом, добиться наилучшего по вашему мнению результата.

Глава 8. Элементы управления

В этой главе мы продемонстрируем вам, как можно сочетать элементы управления Windows, например линейки прокрутки, и звуковую библиотеку фирмы TS. Кроме того, здесь описано несколько новых sp_-функций.

Элементы управления Windows, такие как линейки прокрутки, — очень удобный инструментарий для реализации приложений, использующих звук. Линейкой прокрутки, например, можно воспользоваться для изменения скорости и громкости воспроизведения или позиции начала звукового фрагмента.

Программа Rotate

Мы рекомендуем перед рассмотрением текста программы откомпилировать и запустить ее. Это поможет вам лучше понять принцип ее работы.

Компиляция, компоновка и запуск программы Rotate

Для компиляции и компоновки программы Rotate с помощью компилятора фирмы Microsoft необходимо выполнить следующие действия:

- Убедиться в том, что ваш компьютер находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
NMAKE Rotate.mak    [Enter]
```

Для компиляции и компоновки программы Rotate с помощью компилятора фирмы Borland нужно выполнить такие действия:

- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
MAKE -f Rotate.bmk    [Enter]
```

Для запуска программы Rotate необходимо:

- Выбрать из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- Использовать диалоговое окно **Пролистать** для выбора файла

```
c:\spSDK\Samp4Win\Rotate.exe
```

Появляющееся при запуске главное окно приложения приведено на рис. 8.1.

Главное меню программы Rotate (см. рис. 8.2) состоит из трех команд — **Завершить**, **О программе** и **Вращать**. Если пользователь выбирает команду **Завершить**, то программа произносит фразу "Good-Bye" и завершается. Если он выбирает команду **О программе**, то выводится диалоговое окно **О программе Rotate**, которое изображено на рис. 8.3. Если пользователь выбирает команду **Вращать**, то появляется диалоговое окно, приведенное на рис. 8.4. Это диалоговое окно содержит линейку прокрутки и три командные кнопки — **Играть**, **Игра назад** и **Выход**. Если пользователь нажимает командную кнопку **Выход**, то диалоговое окно закрывается. Если он нажимает командную кнопку **Играть**, то воспроизводится фраза "Press any key to continue". Если пользователь нажимает командную кнопку **Игра назад**, то фраза "Press any key to continue" воспроизводится в обратном порядке ("eunitnoc ot yek yna sserP"). Кроме того, пользователь может изменить скорость воспроизведения посредством изменения позиции бегунка на линейке прокрутки **Скорость**.

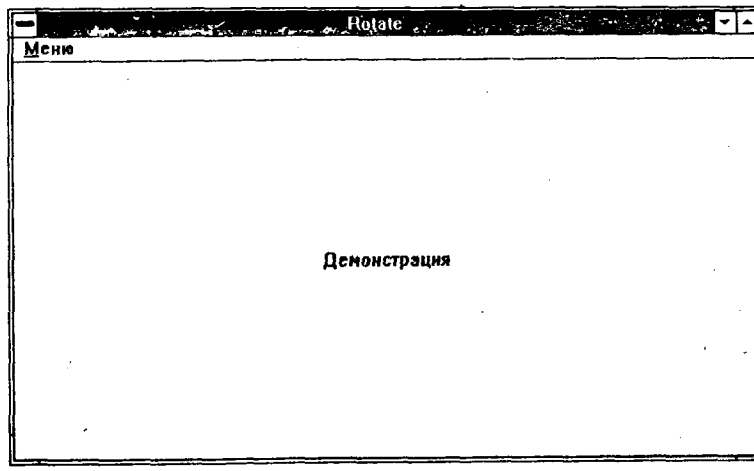


Рис. 8.1. Главное окно программы Rotate

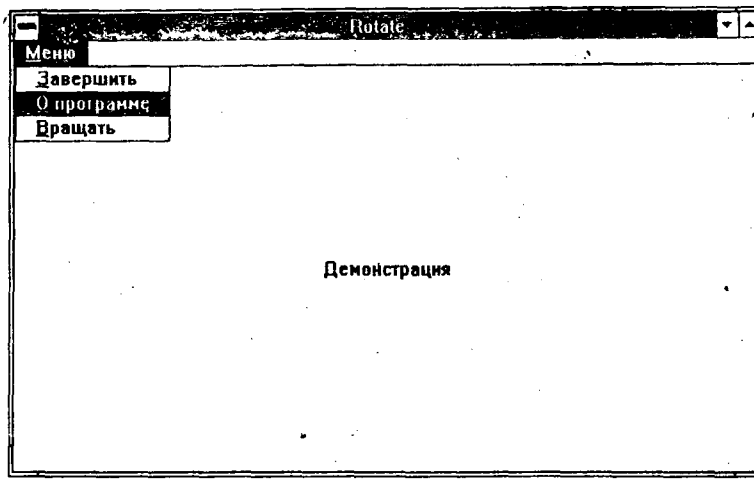


Рис. 8.2. Главное меню программы Rotate

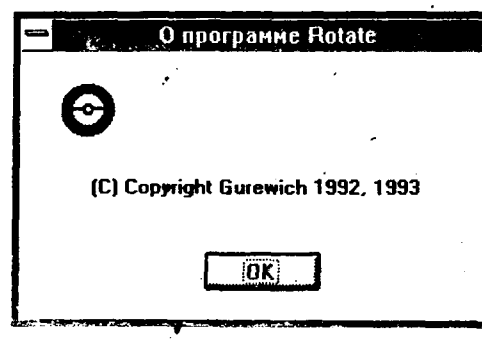


Рис. 8.3. Диалоговое окно О программе Rotate

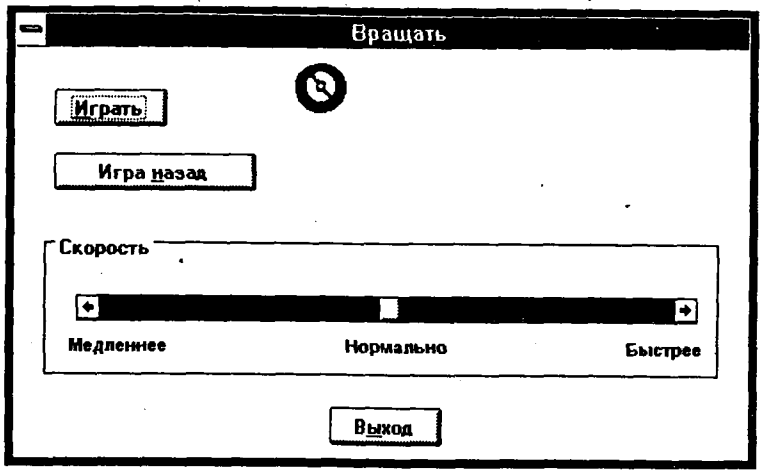


Рис. 8.4. Диалоговое окно Вращать

Файлы, входящие в состав программы Rotate

Полный комплект программы Rotate включает следующие файлы:

Rotate.h	Файл .h
Rotate.c	Текст программы на C
Rotate.rc	Описание ресурсов программы
Rotate.def	Файл определения модуля программы
Rotate.mak	Make-файл программы
Rotate.ico	Пиктограмма программы
Axis1.bmp	Положение #1 диска
Axis2.bmp	Положение #2 диска
Axis3.bmp	Положение #3 диска
Axis4.bmp	Положение #4 диска

Все эти файлы находятся на прилагаемой к книге дискете и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Приведем полный листинг программы Rotate (см. листинги 8.1–8.5).

Листинг 8.1. Rotate.h

```
/*================================================
 Имя файла: rotate.h

 (C) Copyright Gurewich 1992, 1993
 ================================================*/

/*--------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc         ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export AboutDlgProc    ( HWND, UINT, UINT, LONG ) ;
```

```
BOOL FAR PASCAL _export RotateDlgProc  ( HWND, UINT, UINT, LONG ) ;

void ChangeSpeedWasRequested ( HWND );
void RotateRight             ( HWND );
void RotateLeft              ( HWND );
void DisplayAxis             ( HWND, HBITMAP );

/*------
 #define
 -------*/
#define IDM_QUIT  1  /* Команда меню Завершить   */
#define IDM_ABOUT 2  /* Команда меню О программе */
#define IDM_ROTATE 3 /* Команда меню Вращать     */

#define PLAY_PB           101  /* Кнопка Играть.     */
#define SPEED_CHANGE_SB   102  /* Линейка изменения скорости. */
#define EXIT_PB           103  /* Кнопка Выход.      */
#define PLAY_BACKWARD_PB  104  /* Кнопка Игра назад */


/*-------------------------
 Линейка изменения скорости
 -------------------------*/
#define NAT_SPEED_SCROLL    0
#define MIN_SPEED_SCROLL -100
#define MAX_SPEED_SCROLL  100

/*--------------------
 Глобальные переменные
 --------------------*/
char       gszAppName[] = "Rotate" ; /* Имя нашего приложения. */
HINSTANCE  ghInst;                   /* Текущий экземпляр.     */

int        giSpeedScroll;            /* Позиция на линейке прокрутки. */

HBITMAP    ghAxis1;
HBITMAP    ghAxis2;
HBITMAP    ghAxis3;
HBITMAP    ghAxis4;
```

Листинг 8.2. Rotate.c

```
/*==============================================================
 ПРОГРАММА: rotate.c
 ----------
 (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

 ОПИСАНИЕ ПРОГРАММЫ:
 -------------------
 Эта программа основана на Generic 1.

 В программе имеется линейка изменения скорости, которая
 позволяет пользователю изменять скорость воспроизведения.
```

```
 При нажатии кнопки Играть воспроизводится звуковой файл —
 параллельно с выводом на экран изображения вращающегося диска.

 При нажатии кнопки Игра назад звуковой файл воспроизводится
 в обратном порядке, параллельно с выводом на экран изображения
 диска, вращающегося в обратном направлении.
 ================================================================*/
/*-------
 #include
 --------*/
/*-----------------------------------------------
 windows.h  требуется для всех приложений Windows.
 -----------------------------------------------*/
#include <windows.h>

/*-------------------------------------------------------------
 sp4Win.h требуется для того, чтобы функции sp_, макросы SP_
 и #define из библиотеки звуковой поддержки могли быть
 использованы в этом приложении.
 -------------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-------------------------------------------------------------
 Определения и прототипы, специфические для данного приложения.
 -------------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\rotate.h"

/*=================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HWND     hWnd; /* Указатель на окно нашего приложения.          */
MSG      msg;  /* Сообщения, обрабатываемые нашим приложением. */
WNDCLASS wc;   /* Класс окна для нашего приложения.             */

/*-----------------------------------------------
 Создаем экземпляр глобальной переменной hInstance.
 -----------------------------------------------*/
ghInst = hInstance;

/*-----------------------------------------------------------------
 Инициализируем структуру оконного класса и создаем оконный класс.
 -----------------------------------------------------------------*/
if ( !hPrevInstance )
   {
   /*-------------------------------------
   Если условие выполняется, то это самый
   первый запуск нашего приложения.
   -------------------------------------*/
```

```
    wc.style         = CS_HREDRAW | CS_VREDRAW ;
    wc.lpfnWndProc   = WndProc ;
    wc.cbClsExtra    = 0 ;
    wc.cbWndExtra    = 0 ;
    wc.hInstance     = ghInst ;
    wc.hIcon         = LoadIcon   ( ghInst, "IconOfRotate" ) ;
    wc.hCursor       = LoadCursor ( NULL, IDC_ARROW      ) ;
    wc.hbrBackground = GetStockObject ( WHITE_BRUSH );
    wc.lpszMenuName  = gszAppName ;
    wc.lpszClassName = gszAppName ;

   /*-----------------
    Регистрируем окно.
    ------------------*/
    RegisterClass ( &wc );

    }/* конец if(!hPrevInstance) */

 /*------------------------------
  Создаем окно нашего приложения.
  ------------------------------*/
 hWnd = CreateWindow ( gszAppName,
                       gszAppName,
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT,
                       CW_USEDEFAULT,
                       CW_USEDEFAULT,
                       CW_USEDEFAULT,
                       NULL,
                       NULL,
                       ghInst,
                       NULL );

 /*-----------------------------------------
  Выводим и обновляем окно нашего приложения.
  -----------------------------------------*/
 ShowWindow   ( hWnd, nCmdShow );
 UpdateWindow ( hWnd );

 /*-------------------------
  Цикл обработки сообщений.
  -------------------------*/
 while ( GetMessage ( &msg, NULL, 0, 0 ) )
       {
       TranslateMessage ( &msg );
       DispatchMessage  ( &msg );
       }

 return msg.wParam ;

 } /* Конец функции. */
 /*======================== конец WinMain() ====================*/

 /*=================
  ФУНКЦИЯ: WndProc()
  ==================*/
```

```
/*----------------------------
 ОПИСАНИЕ: обработка сообщений.
 ----------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
{
/*-----------------------------------
 Локальные и статистические переменные.
 -----------------------------------*/
static HDC          hdc;      /* Для вывода.   */
static PAINTSTRUCT  ps;       /* Для вывода.   */
RECT                rect;     /* Для вывода текста. */

int         iOpenHelloResult; /* Результат открытия сеанса "Hello". */
int         iOpenPressResult; /* Результат открытия сеанса "Press". */

static FARPROC lpfnAboutDlgProc; /* Для диалогового окна О программе. */
static FARPROC lpfnRotateDlgProc ; /* Для диалогового окна Вращать.   */

switch ( message )
       {
       case WM_CREATE:
            /*----------------------
            Открываем сеанс "Press".
            ----------------------*/
            iOpenPressResult = sp_OpenSession (
                               "c:\\spSDK\\Sfiles\\Press.s",
                               SP_NON_STAND_ALONE,
                               0L,
                               SP_S_TYPE);
            /*-----------------------------------------
            Завершаем работу, если не можем открыть файл.
            -----------------------------------------*/
            if ( iOpenPressResult != SP_NO_ERRORS )
               {
               MessageBox ( NULL,
                            "Невозможно начать работу со звуком.",
                            "Сообщение от rotate.exe",
                            MB_ICONINFORMATION );

            /*------------------------
            Завершаем наше приложение.
            ------------------------*/
            SendMessage ( hWnd, WM_DESTROY, 0 ,0 );
                }

            /*-------------------------------------
            Получаем lpfnAboutDlgPro, указатель на
            диалоговое окно О программе.
            -------------------------------------*/
            lpfnAboutDlgProc =
            MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);
```

```
        /*--------------------------------------
        Получаем lpfnRotateDlgProc, указатель на
        диалоговое окно Вращать.
        --------------------------------------*/
        lpfnRotateDlgProc =
        MakeProcInstance (( FARPROC) RotateDlgProc, ghInst);

        return 0;

    case WM_PAINT:
        hdc = BeginPaint ( hWnd, &ps );
        GetClientRect ( hWnd, &rect );
        DrawText ( hdc,
                   "Демонстрация",
                   -1,
                   &rect,
                   DT_SINGLELINE | DT_CENTER | DT_VCENTER );
                   EndPaint ( hWnd, &ps );
                   return 0;

    case WM_COMMAND:
        /*--------------------
        Обработка команд меню.
        ----------------------*/
        switch (wParam)
               {
               case IDM_QUIT:
                    /*-----------------------------------
                    Пользователь выбрал команду Завершить.
                    -----------------------------------*/
                    DestroyWindow (hWnd);
                    return 0L;

               case IDM_ABOUT:
                    /*-----------------------------------
                    Пользователь выбрал команду О программе.
                    -----------------------------------*/
                    DialogBox ( ghInst,
                                "AboutBox",
                                hWnd,
                                lpfnAboutDlgProc );
                    return 0;

               case IDM_ROTATE :
                    /*-----------------------------------
                    Пользователь выбрал команду Вращать.
                    -----------------------------------*/
                    DialogBox ( ghInst,
                                "RotateBox",
                                hWnd,
                                lpfnRotateDlgProc );
                    return 0;

               }/* конец switch (wParam) */
```

```
      case WM_DESTROY:
           iOpenHelloResult =
           sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                            SP_NON_STAND_ALONE,
                            0L,
                            SP_TS_TYPE);

           /*-----------------------------
           Если сеанс работы со звуком
           не был открыт, ничего не делаем.
           -----------------------------*/

           if ( iOpenHelloResult == SP_NO_ERRORS )
              {
              /*-----------------------------------
              Воспроизводим часть звукового файла.
              Сколько играть: от 30000 до 40000
              Фраза: "Good-Bye"
              -----------------------------------*/
              sp_PlayF ( 30000L, 40000L );
              }

           PostQuitMessage (0);
           return 0;

      }/* конец switch (message) */

/*----------------------------
 Сообщение не было обработано.
 ----------------------------*/

return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}/* конец WndProc() */
/*=================== конец WndProc() ====================*/


/*======================
 ФУНКЦИЯ: AboutDlgProc()
 ======================*/

/*-------------------------------------------------
 ОПИСАНИЕ:
 Это функция обработки диалогового окна О программе.
 -------------------------------------------------*/

BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;
```

```
        case WM_COMMAND :
             switch ( wParam )
                    {
                    case IDOK :
                    case IDCANCEL :
                         EndDialog ( hDlg, 0 );
                         return TRUE;
                    }
        }
return FALSE ;
}/* Конец функции. */
/*=============== конец AboutDlgProc() ==============*/

/*=======================
 ФУНКЦИЯ: RotateDlgProc()
 =======================*/
/*-----------------------------------------
 ОПИСАНИЕ:
 Эта функция реализует диалоговое окно Вращать.
 -----------------------------------------*/
BOOL FAR PASCAL _export RotateDlgProc ( HWND hDlg,
                                        UINT message,
                                        UINT wParam,
                                        LONG lParam )
{
static HDC         hdc;      /* Требуется для вывода.   */
static PAINTSTRUCT ps;       /* Требуется для вывода.   */
static HDC         hMemDC;   /* Требуется для вывода    */
                             /* растровых изображений. */

HWND hWndCtrl;
long lCurrentByte;

switch ( message )
      {
      case WM_INITDIALOG :
           /*-------------------------------------------------
           Задаем минимальное и максимальное значения линейки
           изменения скорости воспроизведения.
           -------------------------------------------------*/
           hWndCtrl = GetDlgItem ( hDlg, SPEED_CHANGE_SB );
           SetScrollRange ( hWndCtrl,
                            SB_CTL,
                            MIN_SPEED_SCROLL,
                            MAX_SPEED_SCROLL,
                            TRUE );

           giSpeedScroll = NAT_SPEED_SCROLL;

           /*-------------------------------------------------------
           Положение бегунка на линейке изменения скорости, которое
           соответствует нормальному воспроизведению звука.
           -------------------------------------------------------*/
           SetScrollPos ( hWndCtrl,
                          SB_CTL,
                          NAT_SPEED_SCROLL,
                          TRUE );
```

```
        ghAxis1 = LoadBitmap ( ghInst, "Axis1" );
        ghAxis2 = LoadBitmap ( ghInst, "Axis2" );
        ghAxis3 = LoadBitmap ( ghInst, "Axis3" );
        ghAxis4 = LoadBitmap ( ghInst, "Axis4" );

        return TRUE;

    case WM_PAINT:

        hdc = BeginPaint ( hDlg, &ps );

        hMemDC = CreateCompatibleDC ( hdc );
        SelectObject ( hMemDC, ghAxis1 );

        BitBlt ( hdc,
                 175,
                 10,
                 32,
                 32,
                 hMemDC,
                 0,
                 0,
                 SRCCOPY );
        DeleteDC ( hMemDC );
        EndPaint ( hDlg, &ps );

        return TRUE;

    case WM_HSCROLL:
        /*-----------------------------------------------
        Пользователь изменил позицию бегунка на линейке.
        -----------------------------------------------*/

        switch ( wParam )
               {
               case SB_PAGEUP:
               case SB_LINEUP:
                    /*-------------------------------------
                    Бегунок на линейке переместился влево.
                    -------------------------------------*/
                    giSpeedScroll--;
                    ChangeSpeedWasRequested ( hDlg );
                    return TRUE;

               case SB_PAGEDOWN:
               case SB_LINEDOWN:
                    /*-------------------------------------
                    Бегунок на линейке переместился вправо.
                    -------------------------------------*/
                    giSpeedScroll++;
                    ChangeSpeedWasRequested ( hDlg );
                    return TRUE;
```

```
            case SB_THUMBPOSITION:
                 /*--------------------------
                 Бегунок на линейке установлен
                 в исходное положение.
                 --------------------------*/
                 giSpeedScroll = LOWORD ( lParam );
                 ChangeSpeedWasRequested ( hDlg );
                 return TRUE;

} /* конец switch(wParam) */

case WM_COMMAND :
     switch ( wParam )
            {
            case PLAY_PB:
                 lCurrentByte = 0L;
                 while ( 1 )
                      {
                      lCurrentByte =
                      sp_PlayF ( lCurrentByte,
                      lCurrentByte + 2000L );

                      RotateRight ( hDlg );

                      if ( lCurrentByte == 0L )
                           break;

                           }/* конец while (1) */

                      return TRUE;

            case PLAY_BACKWARD_PB:
                 lCurrentByte = 16000L;
                 while ( 1 )
                       {
                       lCurrentByte =
                       sp_PlayB ( lCurrentByte,
                       lCurrentByte -2000L );

                      RotateLeft ( hDlg );

                      if ( lCurrentByte == 0L )
                           break;

                           }/* конец of while (1) */

                      return TRUE;

            case EXIT_PB:
            case IDOK :
            case IDCANCEL :
                 /*---------------------------------
                 Устанавливаем TSEngine на нормальную
                 скорость воспроизведения.
                 ---------------------------------*/
```

```
                        sp_SetNewSpeed ( MAX_SPEED_SCROLL,
                                         NAT_SPEED_SCROLL,
                                         NAT_SPEED_SCROLL );

                        DeleteObject ( ghAxis1 );
                        DeleteObject ( ghAxis2 );
                        DeleteObject ( ghAxis3 );
                        DeleteObject ( ghAxis4 );

                        EndDialog ( hDlg, 0 );
                        return TRUE;
                    }/* конец switch(wParam) */

        }/* конец switch(wParam) */

    return FALSE ;

    }/* Конец функции. */
    /*================ конец RotateDlgProc() ==============*/


    /*======
     ФУНКЦИЯ
     =======*/
    /*----------------------------------------
     Пользователь изменил положение бегунка на
     линейке изменения скорости воспроизведения.
    ----------------------------------------*/
    void ChangeSpeedWasRequested ( HWND hWnd )
    {

    HWND  hWndCtrl;

    if ( giSpeedScroll >  MAX_SPEED_SCROLL )
         giSpeedScroll = MAX_SPEED_SCROLL;

    if ( giSpeedScroll <  MIN_SPEED_SCROLL )
         giSpeedScroll = MIN_SPEED_SCROLL;

    sp_SetNewSpeed ( MAX_SPEED_SCROLL,
                     NAT_SPEED_SCROLL,
                     giSpeedScroll );

    hWndCtrl = GetDlgItem ( hWnd, SPEED_CHANGE_SB );
    SetScrollPos ( hWndCtrl, SB_CTL, giSpeedScroll, TRUE );

    }/* Конец функции. */
    /*=================== конец функции ==============*/


    /*=======
     ФУНКЦИЯ
     ========*/
    void RotateRight ( HWND hWnd )
    {
    static int iRotateFlag = 1;
```

```
if ( iRotateFlag == 1 )
   {
   DisplayAxis ( hWnd, ghAxis1 );
   iRotateFlag = 2;
   return;
   }

if ( iRotateFlag == 2 )
   {
   DisplayAxis ( hWnd, ghAxis2 );
   iRotateFlag = 3;
   return;
   }

if ( iRotateFlag == 3 )
   {
   DisplayAxis ( hWnd, ghAxis3 );
   iRotateFlag = 4;
   return;
   }

if ( iRotateFlag == 4 )
   {
   DisplayAxis ( hWnd, ghAxis4 );
   iRotateFlag = 1;
   return;
   }

}/* Конец функции. */
/*============ конец функции RotateRight() ===========*/


/*=======
 ФУНКЦИЯ
 =======*/
void RotateLeft ( HWND hWnd )
{
static int iRotateFlag = 1;

if ( iRotateFlag == 1 )
   {
   DisplayAxis ( hWnd, ghAxis1 );
   iRotateFlag = 4;
   return;
   }

if ( iRotateFlag == 4 )
   {
   DisplayAxis ( hWnd, ghAxis4 );
   iRotateFlag = 3;
   return;
   }

if ( iRotateFlag == 3 )
   {
   DisplayAxis ( hWnd, ghAxis3 );
```

```
   iRotateFlag = 2;
   return;
   }

if ( iRotateFlag == 2 )
   {
   DisplayAxis ( hWnd, ghAxis2 );
   iRotateFlag = 1;
   return;
   }

}/* Конец функции. */
/*============ конец функции RotateLeft() ===========*/


/*=======
 ФУНКЦИЯ
 =======*/
void DisplayAxis ( HWND hWnd, HBITMAP hAxis)
{
static HDC hdc;
static HDC hMemDC;

hdc = GetDC ( hWnd );
hMemDC = CreateCompatibleDC ( hdc );
SelectObject ( hMemDC, hAxis );

BitBlt ( hdc,
         175,
         10,
         32,
         32,
         hMemDC,
         0,
         0,
         SRCCOPY );

DeleteDC ( hMemDC );
ReleaseDC ( hWnd, hdc   );

}/* Конец функции. */
/*============ конец функции ===========*/
```

Листинг 8.3. Rotate.rc

```
/*=========================================
 ИМЯ ФАЙЛА: rotate.rc
 ----------
 ОПИСАНИЕ ФАЙЛА:
 ---------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 =========================================*/
```

```
/*-------
 #include
 --------*/
#include <windows.h>
#include "Rotate.h"

/*---
 Menu
 ----*/
Rotate  MENU
BEGIN
   POPUP  "&Меню"
       BEGIN
           MENUITEM "&Завершить", IDM_QUIT
           MENUITEM "&О программе", IDM_ABOUT
           MENUITEM "&Вращать", IDM_ROTATE
       END
END

IconOfRotate ICON Rotate.ico

/*---------------
 Растровые файлы.
-----------------*/
Axis1 BITMAP Axis1.bmp
Axis2 BITMAP Axis2.bmp
Axis3 BITMAP Axis3.bmp
Axis4 BITMAP Axis4.bmp

/*---------------------------
 Диалоговое окно О программе.
-----------------------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "О программе Rotate"
/* необходим локализованный шрифт MS Sans Serif */
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON    "OK", IDOK, 64, 75, 40, 14
    CTEXT         "(C) Copyright Gurewich 1992, 1993",
                  -1, 13, 47, 137, 18
    ICON          "IconOfRotate", -1, 14, 12, 18, 20
END

/*-----------------------
 Диалоговое окно Вращать.
-------------------------*/
RotateBox DIALOG 33, 28, 263, 157
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Вращать"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON    "&Играть", PLAY_PB, 14, 17, 40, 14
    SCROLLBAR     SPEED_CHANGE_SB, 22, 96, 222, 10
    PUSHBUTTON    "В&ыход", EXIT_PB, 113, 138, 40, 14
```

```
    GROUPBOX        "Скорость", -1, 11, 72, 241, 54
    LTEXT           "Нормально", -1, 122, 112, 27, 8
    LTEXT           "Быстрее", -1, 222, 112, 20, 8
    LTEXT           "Медленнее", -1, 19, 112, 20, 8
    PUSHBUTTON      "Игра &назад", PLAY_BACKWARD_PB,
                    14, 42, 72, 14
END
```

Листинг 8.4. Rotate.def

```
;=========================================
; Файл определения модуля программы Rotate.c
;=========================================

NAME        Rotate

DESCRIPTION  'The Rotate program. (C) Copyright Gurewich 1992, 1993'

EXETYPE     WINDOWS

STUB        'WINSTUB.EXE'

CODE  PRELOAD MOVEABLE DISCARDABLE

DATA  PRELOAD MOVEABLE MULTIPLE

HEAPSIZE    1024
STACKSIZE   8192
```

Листинг 8.5. Rotate.mak

```
#==========
# Rotate.mak
#==========

Rotate.exe : Rotate.obj Rotate.h Rotate.def Rotate.res
    link /nod Rotate.obj, Rotate.exe, NUL, \
       slibcew.lib oldnames.lib libw.lib commdlg \
       c:\spSDK\TegoWlib\TegoWin.lib, \
       Rotate.def
    rc -t Rotate.res

Rotate.obj : Rotate.c Rotate.h
   cl -c -G2sw -Ow -W3 -Zp Rotate.c

Rotate.res : Rotate.rc Rotate.h Rotate.ico
   rc -r Rotate.rc
```

Открытие сеанса работы со звуком

В варианте WM_CREATE мы открываем сеанс работы со звуковым файлом типа S C:\spSDK\SFiles\Press.s, чтобы создать неавтономное приложение:

```
application:
switch ( message )
       {
       case WM_CREATE:

            /----------------------
            Открываем сеанс "Press".
            ----------------------*/
            iOpenPressResult = sp_OpenSession (
                               "c:\\spSDK\\Sfiles\\Press.s",
                               SP_NON_STAND_ALONE,
                               0L,
                               SP_S_TYPE);
            /*-----------------------------------------
            Завершаем работу, если не можем открыть файл.
            -----------------------------------------*/
            if ( iOpenPressResult != SP_NO_ERRORS )
               {
               MessageBox ( NULL,
                            "Невозможно начать работу со звуком.",
                            "Сообщение от rotate.exe",
                            MB_ICONINFORMATION );

               /*-------------------------
               Завершаем наше приложение.
               -------------------------*/
               SendMessage ( hWnd, WM_DESTROY, 0 ,0 );
               }
            .........................
            .........................
            .........................
            return 0;
```

Отображение диалогового окна Вращать

IDM_CREATE соответствует ситуации, когда пользователь выбирает из главного меню программы команду **Вращать**. В этом варианте мы вызываем функцию DialogBox(), которая отображает диалоговое окно **Вращать**.

```
case IDM_ROTATE :
  /*---------------------------------
  Пользователь выбрал команду Вращать.
  ---------------------------------*/
  DialogBox ( ghInst,
              "RotateBox",
              hWnd,
              lpfnRotateDlgProc );
  return 0;
```

Инициализация диалогового окна Вращать

Диалоговое окно **Вращать** инициализируем в варианте WM_INITDIALOG в функции диалогового окна **Вращать**:

```
case WM_INITDIALOG:
     ..........................................................
     .... Установим максимальное и минимальное ..............
     .....значения на линейке изменения скорости .............
     ..........................................................
```

```
        ..........................................................
        .... Установим бегунок на линейке изменения скорости .....
        .... в значение нормальной скорости воспроизведения  .....
        ..........................................................

        ..........................................................
        .... Сохраним значение, соответствующее текущему .........
        .... положению бегунка на линейке изменения скорости .....
        ..........................................................

        ..........................................................
        .... Загрузим растровые файлы, которые  ..................
        .... используются при выводе вращающегося колеса .........
        ..........................................................
return TRUE;
```

Установка минимального и максимального значений на линейке изменения скорости

Линейка изменения скорости определена в файле Rotate.h, а местоположение ее изображения в диалоговом окне **Вращать** — в файле Rotate.rc. Кроме того, мы определили в Rotate.h три константы:

```
#define NAT_SPEED_SCROLL    0
#define MIN_SPEED_SCROLL -100
#define MAX_SPEED_SCROLL  100
```

Эти константы определяют нормальную скорость воспроизведения как 0, максимальную скорость — как 100, а минимальную — как –100. Градуировка шкалы на 200 делений позволяет изменять скорость воспроизведения очень плавно.

Максимальное и минимальное значения на линейке изменения скорости устанавливаются следующим образом. Мы получаем значение hWndCtrl — указателя на линейку изменения скорости, после чего вызываем функцию SetScrollRange():

```
/*-----------------------------------------------
 Задаем минимальное и максимальное значения для
 линейки изменения скорости воспроизведения.
 -----------------------------------------------*/
 hWndCtrl = GetDlgItem ( hDlg, SPEED_CHANGE_SB );
 SetScrollRange ( hWndCtrl,
                  SB_CTL,
                  MIN_SPEED_SCROLL,
                  MAX_SPEED_SCROLL,
                  TRUE );
```

Установка бегунка на нормальную скорость воспроизведения

Исходное положение бегунка линейки изменения скорости — середина шкалы. Очевидно, что это положение соответствует нормальной скорости воспроизведения. Для установки бегунка на середину используем функцию SetScrollPos():

```
/*-----------------------------------------------
 Позиция бегунка на линейке изменения скорости
 для нормального воспроизведения звука.
 -----------------------------------------------*/
 SetScrollPos ( hWndCtrl,
                SB_CTL,
                NAT_SPEED_SCROLL,
                TRUE );
```

Сохранение текущего положения бегунка линейки изменения скорости воспроизведения

Во время работы программы мы отслеживаем положение бегунка линейки изменения скорости воспроизведения, сохраняя соответствующее значение в переменной giSpeedScroll. Например, в случае, когда бегунок установлен на нормальную скорость воспроизведения, мы инициализируем переменную giSpeedScroll следующим значением:

```
giSpeedScroll = NAT_SPEED_SCROLL;
```

Загрузка растровых файлов, используемых в диалоговом окне Вращать

В диалоговом окне **Вращать** используются четыре растровых файла. Каждый из них отображает диск в различном положении. Эти файлы приведены на рис. 8.5. Мы загружаем растровые файлы с помощью функции LoadBitmap():

```
ghAxis1 = LoadBitmap ( ghInst, "Axis1" );
ghAxis2 = LoadBitmap ( ghInst, "Axis2" );
ghAxis3 = LoadBitmap ( ghInst, "Axis3" );
ghAxis4 = LoadBitmap ( ghInst, "Axis4" );
```

Рис. 8.5. Растровые рисунки, используемые в диалоговом окне Вращать

Вариант WM_PAINT для диалогового окна Вращать

В варианте WM_PAINT для диалогового окна **Вращать** мы выводим изображение диска в положении Axis1. Делаем это с помощью функции BitBlt():

```
case WM_PAINT:
     hdc = BeginPaint ( hDlg, &ps );

     hMemDC = CreateCompatibleDC ( hdc );
     SelectObject ( hMemDC, ghAxis1 );

     BitBlt ( hdc,
              175,
              10,
              32,
              32,
              hMemDC,
              0,
              0,
              SRCCOPY );
     DeleteDC ( hMemDC );
     EndPaint ( hDlg, &ps );

     return TRUE;
```

Вариант WM_HSCROLL

Всякий раз, когда пользователь изменяет положение бегунка — увеличивая или уменьшая скорость воспроизведения — генерируется сообщение WM_HSCROLL:

```
case WM_HSCROLL:
     /*-------------------------------------------------
     Пользователь изменил положение бегунка на линейке.
     -------------------------------------------------*/

     switch ( wParam )
            {
            case SB_PAGEUP:
            case SB_LINEUP:
                 /*------------------
                 Бегунок на линейке
                 переместился влево.
                 ------------------*/
                 giSpeedScroll--;
                 ChangeSpeedWasRequested ( hDlg );
                 return TRUE;

            case SB_PAGEDOWN:
            case SB_LINEDOWN:
                 /*------------------
                 Бегунок на линейке
                 переместился вправо.
                 ------------------*/
                 giSpeedScroll++;
                 ChangeSpeedWasRequested ( hDlg );
                 return TRUE;

            case SB_THUMBPOSITION:
                 /*------------------
                 Бегунок передвинут.
                 ------------------*/
                 giSpeedScroll = LOWORD ( lParam );
                 ChangeSpeedWasRequested ( hDlg );
                 return TRUE;

            } /* конец switch(wParam) */
```

Уменьшение и увеличение скорости воспроизведения

Если wParam присвоено значение SB_PAGEUP или SB_LINEUP, то это означает, что пользователь передвинул бегунок линейки изменения скорости воспроизведения на одно деление в сторону уменьшения скорости. В таком случае мы уменьшаем значение переменной giSpeedScroll на 1, а затем вызываем функцию ChangeSpeedWasRequested().

Обработка перемещения бегунка в сторону увеличения скорости воспроизведения идентична обработке в случае уменьшения скорости, отличие состоит лишь в том, что значение переменной giSpeedScroll увеличивается.

Перемещение бегунка линейки изменения скорости воспроизведения

Если пользователь передвигает бегунок линейки изменения скорости воспроизведения, то генерируется сообщение SP_THUMBPOSITION. В варианте получения сообщения SP_THUMBPOSITION мы присваиваем переменной giSpeedScroll значение,

соответствующее новой позиции бегунка линейки. Это значение передается в младшем слове переменной wParam данного сообщения. И последнее, что мы делаем в этом варианте, — это вызываем функцию ChangeSpeedWasRequested().

Функция ChangeSpeedWasRequested()

Мы вызываем функцию ChangeSpeedWasRequested() при каждом изменении положения бегунка линейки изменения скорости воспроизведения. Эта функция сравнивает новое значение giSpeedScroll с граничными значениями:

```
if ( giSpeedScroll >  MAX_SPEED_SCROLL )
     giSpeedScroll = MAX_SPEED_SCROLL;

if ( giSpeedScroll <  MIN_SPEED_SCROLL )
     giSpeedScroll = MIN_SPEED_SCROLL;
```

После изменения giSpeedScroll мы вызываем TSEngine для изменения скорости воспроизведения посредством вызова функции sp_SetNewSpeed():

```
sp_SetNewSpeed ( MAX_SPEED_SCROLL,
                 NAT_SPEED_SCROLL,
                 giSpeedScroll );
```

После этого присваиваем указателю соответствующее новое значение:

```
hWndCtrl = GetDlgItem ( hWnd, SPEED_CHANGE_SB );
SetScrollPos ( hWndCtrl, SB_CTL, giSpeedScroll, TRUE );
```

Функция sp_SetNewSpeed()

Функция sp_SetNewSpeed() указывает TS Engine на то, что необходимо установить новую скорость воспроизведения. Прототип функции sp_SetNewSpeed() объявлен в файле c:\spSDK\TegoWlib.h:

```
int sp_SetNewSpeed( int, int, int );
```

Функция возвращает целое, но это значение не используется. Функции передаются три параметра. Первый и второй параметры — это целые, которые задают соответственно максимальную и нормальную скорости воспроизведения; третий параметр задает ту скорость, которую нужно установить.

Например, вызов

```
sp_SetNewSpeed ( 10, 5, 6 );
```

означает, что максимальная скорость воспроизведения равна 10, нормальная — 5, а установить надо 6, что больше пяти, и, следовательно, воспроизведение будет производиться быстрее.

В нашей программе мы вызываем sp_SetNewSpeed() следующим образом:

```
sp_SetNewSpee ( MAX_SPEED_SCROLL,
                NAT_SPEED_SCROLL,
                giSpeedScroll);
```

Это означает, что максимальная скорость будет равна константе MAX_SPEED_SCROLL (которую мы определили равной 100), нормальная — константе NAT_SPEED_SCROLL (которую мы определили равной 0), а новая скорость задается giSpeedScroll (которая может принимать значения от –100 до 100).

Примечание:

Функции sp_SetNewSpeed() передаются три параметра:

Первый параметр: Целое. Задает максимальную скорость.

Второй параметр: Целое. Задает нормальную скорость.

Третий параметр: Целое. Задает новую скорость.

Перед вызовом функции sp_SetNewSpeed() уже должна быть вызвана функция sp_OpenSession().

Командная кнопка PLAY_PB диалогового окна Вращать

Всякий раз, когда пользователь нажимает командную кнопку **Играть**, выполняется группа операторов, обрабатывающих вариант PLAY_PB. В варианте PLAY_PB мы воспроизводим звуковой файл фрагментами по 2000 байт каждый. Между воспроизведением фрагментов вызываем функцию RotateRight():

```
case PLAY_PB:
     lCurrentByte = 0L;
     while ( 1 )
           {
           lCurrentByte =
           sp_PlayF ( lCurrentByte,
                      lCurrentByte + 2000L );

           RotateRight ( hDlg );

           if ( lCurrentByte == 0L )
                break;

           }/* конец while (1) */

           return TRUE;
```

Функция RotateRight() осуществляет вращение диска по часовой стрелке.

Вариант PLAY_BACKWARD_PB диалогового окна Вращать

Всякий раз, когда пользователь нажимает командную кнопку **Обратная игра**, выполняется группа операторов, обрабатывающих вариант PLAY_BACKWARD_PB:

```
case PLAY_BACKWARD_PB:
     lCurrentByte = 16000L;
     while ( 1 )
           {
           lCurrentByte =
           sp_PlayB ( lCurrentByte,
                      lCurrentByte - 2000L );

           RotateLeft ( hDlg );

           if ( lCurrentByte == 0L )
                break;

           }/* конец of while (1) */
      return TRUE;
```

Функция sp_PlayB()

Функция sp_PlayB() — функция C из звуковой библиотеки фирмы TS (вы могли определить это по первым трем символам названия — sp_). Прототип этой функции объявлен в файле c:\spSDK\TegoWlib\sp4Win.h:

```
long sp_PlayB ( long lStartPoint,
                long lEndPoint );
```

Функция sp_PlayB() очень похожа на функцию sp_PlayF() (F означает "Forward" — "вперед", а B — "Backward" — "назад").

Функция возвращает длинное целое, которое указывает на последний воспроизведенный байт. Первый передаваемый параметр — это длинное целое, указывающее на начало воспроизведения, второй — длинное целое, указывающее на конец воспроизведения. Поскольку функция sp_PlayB() воспроизводит звуковой файл в обратном направлении, то первый параметр должен быть больше второго. В противном случае результаты выполнения функций sp_PlayF() и sp_PlayB() будут идентичными.

Примечание:

Функции sp_PLayB() передаются два параметра:

Первый параметр: Длинное целое. Указывает позицию байта, с которого начинается воспроизведение фрагмента.

Второй параметр: Длинное целое. Указывает позицию байта, на котором воспроизведение фрагмента оканчивается.

Возвращаемое значение: Длинное целое, которое указывает на последний воспроизведенный байт.

До того как функция sp_PlayB() будет вызвана, должна быть вызвана функция sp_OpenSession().

Поскольку функция sp_PlayB() воспроизводит файл в обратном направлении, то значение первого передаваемого параметра должно быть больше значения второго параметра.

Функции RotateRight() и RotateLeft()

Функция RotateRight() реализует вращение диска по часовой стрелке, а функция RotateLeft() — против часовой стрелки.

Функция RotateRight() вызывается в промежутках между воспроизведением фрагментов по 2000 байт в естественном направлении, а функция RotateLeft() — в промежутках между воспроизведением таких фрагментов по 2000 байт в обратном направлении.

В функции RotateRight() используется статическая переменная iRotateFlag. Этот флаг определяет, какое именно состояние диска надо показать в следующий раз. Первоначально флаг установлен в 1. При вызове функции RotateRight() удовлетворяется условие первого оператора if, так как iRotateFlag равен 1, и вызывается функция DisplayAxis():

```
if ( iRotateFlag == 1 )
   {
   DisplayAxis ( hWnd, ghAxis1 );
   iRotateFlag = 2;
   return;
   }
```

В качестве второго параметра функции DisplayAxis () передается переменная ghAxis1, которая задает растровый рисунок диска в положении 1.

В этом же операторе if устанавливаем iRotateFlag равным 2, чтобы при следующем вхождении в функцию удовлетворялось условие второго оператора if:

```
if ( iRotateFlag == 2 )
   {
   DisplayAxis ( hWnd, ghAxis2 );
   iRotateFlag = 3;
   return;
   }
```

На этот раз в качестве второго параметра функции DisplayAxis () передается переменная ghAxis2, задающая растровый рисунок диска в положении 2.

Флаг iRotateFlag определен как статическая переменная, и поэтому его содержимое не теряется при выходе из функции.

Флаг iRotateFlag последовательно принимает следующие значения:

```
1 -> 2 -> 3 -> 4 -> 1 -> 2 -> ...
```

Таким образом создается впечатление, что диск вращается.

Функция RotateLeft() практически идентична RotateRight (), с тем отличием, что флаг iRotateFlag изменяется в другой последовательности:

```
1 -> 4 -> 3 -> 2 -> 1 -> 4 -> ...
```

Это создает иллюзию вращения диска против часовой стрелки.

Дополнительная информация о функции sp_PlayB()

В программе Rotate мы используем функцию sp_PlayB() для воспроизведения в обратном направлении звукового файла типа S. Функцию sp_PlayB() можно использовать и для файлов других типов (например, TS, WAV, VOS, SND). Однако сокращенная версия библиотеки TS поддерживает функцию sp_PlayB() только для файлов типа S.

Существует множество вариантов практического применения функции sp_PlayB(). Например, вы можете написать программу, которая будет поддерживать функции всех кнопок обычного магнитофона: **Пауза**, **Перемотка назад**, **Перемотка вперед** и **Воспроизведение**. При реализации кнопки **Перемотка назад** ваша программа должна будет имитировать перемотку пленки, причем во время этой перемотки она сможет дополнительно воспроизводить звуковой файл в обратном направлении. Кроме того, функцию sp_PlayB() можно использовать для проверки, не содержатся ли в записях скрытые сообщения. По слухам, в некоторых рок-н-ролльных записях в конце песен содержатся сообщения-перевертыши. Вы сможете проверить (или попытаться опровергнуть) эти слухи, используя функцию sp_PlayB() для воспроизведения подозрительных фрагментов в обратном направлении (при таком воспроизведении записи-перевертыши будут звучать, как обычные).

Завершение диалогового окна Вращать

Если пользователь завершает работу с диалоговым окном **Вращать**, мы прибегаем к вызову функции sp_SetNewSpeed() для того, чтобы задать библиотеке TSEngine воспроизведение с нормальной скоростью. Кроме того, мы удаляем растровые рисунки, которые использовались в диалоговом окне **Вращать**:

```
case EXIT_PB:
case IDOK :
case IDCANCEL :
     /*---------------------------------
     Устанавливаем TSEngine на нормальную
     скорость воспроизведения.
     ---------------------------------*/
     sp_SetNewSpeed ( MAX_SPEED_SCROLL,
                      NAT_SPEED_SCROLL,
                      NAT_SPEED_SCROLL );

     DeleteObject ( ghAxis1 );
     DeleteObject ( ghAxis2 );
     DeleteObject ( ghAxis3 );
     DeleteObject ( ghAxis4 );
     EndDialog ( hDlg, 0 );
     return TRUE;
```

Необходимость установки TSEngine на нормальную скорость воспроизведения обусловлена тем, что пользователь мог изменить положение бегунка линейки изменения скорости и, завершая диалог, не вернуть бегунок в исходное положение. Мы хотим быть уверенными в том, что после нажатия командной кнопки **Выход** и завершения работы с диалоговым окном TSEngine будет установлена на нормальную скорость воспроизведения.

Программа Controls

Программа Controls воспроизводит звуковые файлы и предоставляет возможность изменять параметры воспроизведения. Пользователь может изменять скорость воспроизведения (даже во время "проигрывания" файла), перематывать запись вперед или назад, воспроизводить ее в естественном и обратном направлениях. В программе используются несколько новых sp_-функций.

Рекомендуем перед рассмотрением текста программы откомпилировать и запустить ее. Это поможет вам более полно понять ее работу.

Компиляция, компоновка и запуск программы Controls

Для компиляции и компоновки программы Controls с помощью компилятора фирмы Microsoft необходимо выполнить следующие действия:

- Убедиться в том, что компьютер находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
NMAKE Controls.mak    [Enter]
```

Для компиляции и компоновки программы Controls с помощью компилятора фирмы Borland нужно выполнить такие действия:

- Войти в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS ввести:

```
MAKE -f Controls.bmk    [Enter]
```

Для запуска программы Controls необходимо:

- Выбрать из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- Использовать диалоговое окно **Пролистать** для выбора файла

```
c:\spSDK\Samp4Win\Controls.exe
```

Возникающее при запуске главное окно приложения приведено на рис. 8.6.

Главное меню программы Controls (см. рис. 8.7) состоит из трех команд — **Завершить**, **О программе** и **Управление**.

Если пользователь выбирает команду **Завершить**, то программа произносит фразу "Good-Bye" и завершает свою работу. Если он выбирает команду **О программе**, то выводится диалоговое окно **О программе Controls**, которое изображено на рис. 8.8. Если пользователь выбирает команду **Управление**, то отображается диалоговое окно, приведенное на рис. 8.9. В этом окне находятся несколько командных кнопок и линеек прокрутки.

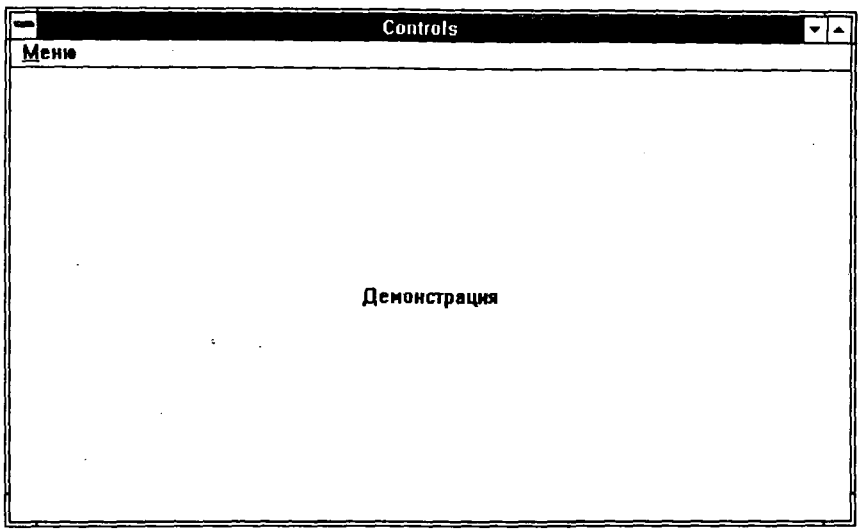


Рис. 8.6. Главное окно программы Controls

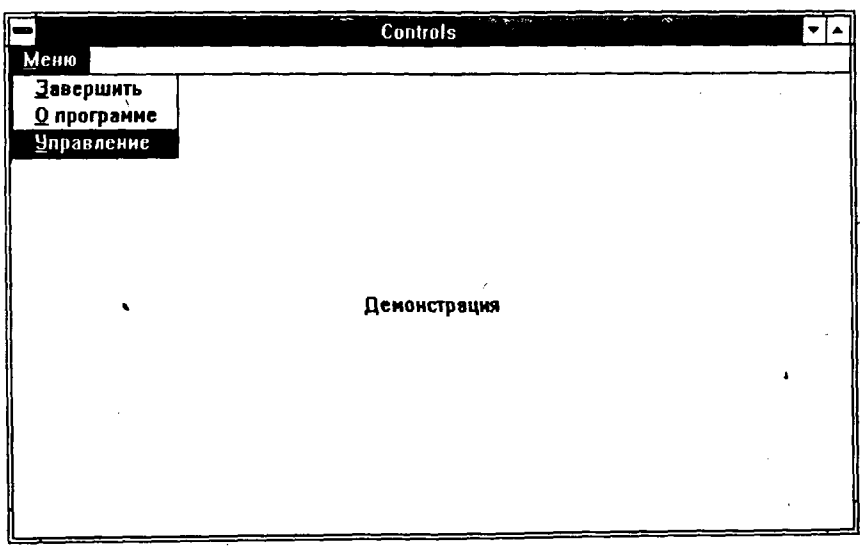


Рис. 8.7. Главное меню программы Controls

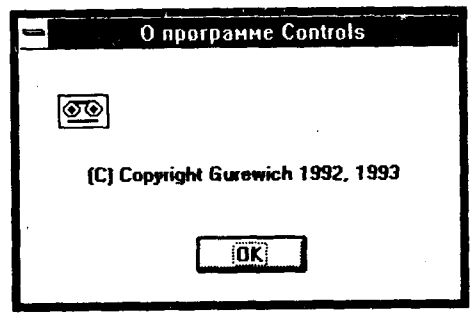


Рис. 8.8. Диалоговое окно О программе Controls

Если пользователь нажимает командную кнопку **Играть**, то воспроизводится звуковой файл. Во время воспроизведения звукового файла пользователь имеет возможность изменять скорость воспроизведения, нажимая клавишу [Б] для увеличения скорости или клавишу [М] — для ее уменьшения. Кроме того, он может нажать клавишу [S] для остановки воспроизведения.

Некоторые элементы диалогового окна во время воспроизведения изменяют свое состояние. Так, число в поле **Позиция в секундах** показывает текущую позицию воспроизведения. Кроме того, диалоговое окно **Управление** содержит линейку прокрутки **вперед/назад**, и на этой линейке также отображается текущая позиция воспроизведения: ее бегунок во время воспроизведения перемещается.

В то время когда файл не воспроизводится, пользователь может изменить положение указателя на линейке прокрутки и таким образом перемотать запись вперед или назад.

В текстовом поле, расположенном в левом верхнем углу диалогового окна, отображается частота дискретизации. В текстовом поле справа отображается длительность звучания файла в секундах.

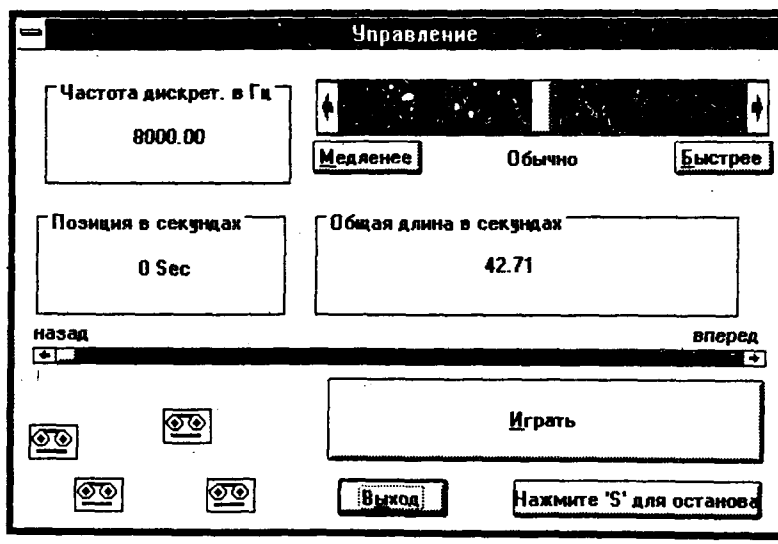


Рис. 8.9. Диалоговое окно Управление

Файлы, входящие в состав программы Controls

Полный комплект программы Controls включает следующие файлы:

Controls.h	Файл .h
Controls.c	Текст программы на C
Controls.rc	Описание ресурсов программы
Controls.def	Файл определения модуля программы
Controls.mak	Make-файл программы
Tape.ico	Пиктограмма программы

Все эти файлы находятся на прилагаемой дискете и установлены на вашем винчестере в директории c:\spSDK\Samp4Win\.

Ниже приведен полный листинг программы Controls (см. листинги 8.6–8.10).

Листинг 8.6. Controls.h

```
/*==========================================
 ИМЯ ФАЙЛА: Controls.h

 (C) Copyright Gurewich 1992, 1993
 ==========================================*/

/*--------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc          ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export AboutDlgProc     ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export ControlsDlgProc  ( HWND, UINT, UINT, LONG ) ;

void ChangeSpeedWasRequested    ( HWND );
void ChangePositionWasRequested ( HWND );
void DisplayLocation            ( HWND, long );
void DisplayFileSize            ( HWND );
void DisplaySamplingRate        ( HWND hDlg);

/*--------
 #define
 -------*/
#define IDM_QUIT      1  /* Команда меню Завершить    */
#define IDM_ABOUT     2  /* Команда меню О программе  */
#define IDM_CONTROLS  3  /* Команда меню Управление   */

#define PLAY_PB        101  /* Кнопка Играть.          */
#define SPEED_SB       102  /* Изменение на линейке.   */
#define POSITION_SB    103  /* Изменение на линейке.   */
#define POSITION_TEXT  104  /* Позиция текста.         */
#define SAMPLING_TEXT  105  /* Собственно текст.       */
#define TOTAL_TEXT     106  /* Общая длина текста.     */
#define FAST_PB        107  /* Кнопка Быстрее.    */
#define SLOW_PB        108  /* Кнопка Медленнее.  */
#define EXIT_PB        109  /* Кнопка Выход.      */
```

```
/*--------------------------
 Линейка изменения скорости.
 --------------------------*/
#define NAT_SPEED_SCROLL    0
#define MIN_SPEED_SCROLL -100
#define MAX_SPEED_SCROLL  100

/*---------------------------
 Положение бегунка на линейке.
 ---------------------------*/
#define MIN_POSITION_SCROLL 0
#define MAX_POSITION_SCROLL 100

/*---------------------
 Глобальные переменные.
 ---------------------*/
char       gszAppName[] = "Controls" ; /* Имя нашего приложения. */
HINSTANCE  ghInst;                     /* Текущий экземпляр.     */
int        giSpeedScroll;     /* Положение бегунка на линейке.*/
int        giPositionScroll;  /* Положение бегунка на линейке.*/
long       glFileSizeInBytes; /* Размер звукового файла.      */
long       glCurrentByte;     /* Позиция в звуковом файле.    */
long       glSamplingRate;
```

Листинг 8.7. Controls.c

```
/*=========================================================================
 ПРОГРАММА: controls.c
 ----------
 (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

 ОПИСАНИЕ ПРОГРАММЫ:
 -------------------
 Эта программа основана на Generic 1.

 Программа имеет линейку прокрутки, которая позволяет
 пользователю изменять скорость воспроизведения и
 позицию в звуковом файле.

 При нажатии командной кнопки Играть воспроизводится
 звуковой файл параллельно с отображением позиции
 в звуковом файле.

 Кроме того, в этой программе показано, как можно
 получать данные о звуковом файле.
 =========================================================================*/

/*--------
 #include
 --------*/
/*-----------------------------------------------
 windows.h  требуется для всех приложений Windows.
 -----------------------------------------------*/
#include <windows.h>
```

```
/*-------------------------------------------------------------
 sp4Win.h требуется для того, чтобы в этом приложении могли
 быть использованы функции sp_, а также макросы SP_ и #define
 из библиотеки звуковой поддержки.
 -------------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-------------------------------------------------------------
 Определения и прототипы, специфические для данного приложения.
 -------------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\Controls.h"

/*-----------------------------------------------------
 Стандартные файлы, которые нужны для работы с файлом.
 -----------------------------------------------------*/
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/*=================
 ФУНКЦИЯ: WinMain()
 =================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HWND     hWnd; /* Идентификатор окна нашего приложения.         */
MSG      msg;  /* Сообщения, обрабатываемые нашим приложением. */
WNDCLASS wc;   /* Класс окна для нашего приложения.             */

/*-----------------------------------------------
 Создаем экземпляр глобальной переменной hInstance.
 -----------------------------------------------*/
ghInst = hInstance;

/*-------------------------------------------------------
 Инициализируем структуру класса окна и создаем этот класс.
 -------------------------------------------------------*/
if ( !hPrevInstance )
   {
   /*-----------------------------------
   Если условие выполняется, то это самый
   первый запуск нашего приложения.
   -----------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfTape" ) ;
```

```
    wc.hCursor       = LoadCursor ( NULL, IDC_ARROW ) ;
    wc.hbrBackground = GetStockObject ( WHITE_BRUSH ) ;
    wc.lpszMenuName  = gszAppName ;
    wc.lpszClassName = gszAppName ;

    /*----------------
    Регистрируем окно.
    ------------------*/
    RegisterClass ( &wc );

    }/* конец if(!hPrevInstance) */
  /*-----------------------------
   Создаем окно нашего приложения.
   -------------------------------*/
  hWnd = CreateWindow ( gszAppName,
                        gszAppName,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        NULL,
                        NULL,
                        ghInst,
                        NULL );

  /*-----------------------------------------
   Выводим и обновляем окно нашего приложения.
   -----------------------------------------*/
  ShowWindow   ( hWnd, nCmdShow );
  UpdateWindow ( hWnd );

  /*-------------------------
   Цикл обработки сообщений.
   -------------------------*/
  while ( GetMessage ( &msg, NULL, 0, 0 ) )
        {
        TranslateMessage ( &msg );
        DispatchMessage  ( &msg );
        }

  return msg.wParam ;
  } /* Конец функции. */
  /*========================= конец WinMain() ====================*/

  /*==================
   ФУНКЦИЯ: WndProc()
   ==================*/
  /*-----------------------------
   ОПИСАНИЕ: обработка сообщений.
   -----------------------------*/

  long FAR PASCAL _export WndProc ( HWND hWnd,
                                    UINT message,
                                    UINT wParam,
                                    LONG lParam )
```

```
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
static HDC  hdc;            /* Для вывода. */
static PAINTSTRUCT ps;      /* Для вывода. */
RECT               rect;    /* Для вывода текста. */

int         iOpenHelloResult; /* Результат открытия сеанса "Hello". */
int         iOpenMusicResult; /* Результат открытия сеанса "Music". */

static FARPROC lpfnAboutDlgProc; /* Для диалогового окна О программе. */
static FARPROC lpfnControlsDlgProc;/* Для диалогового окна Управление.*/

switch ( message )
       {
       case WM_CREATE:

            /*----------------------
            Открываем сеанс "Music".
            ----------------------*/
            iOpenMusicResult = sp_OpenSession (
                               "c:\\spSDK\\TSfiles\\Music.ts",
                               SP_NON_STAND_ALONE,
                               0L,
                               SP_TS_TYPE);
            /*-------------------------------------------
            Завершаем работу, если не можем открыть файл.
            -------------------------------------------*/
            if ( iOpenMusicResult != SP_NO_ERRORS )
               {
               MessageBox ( NULL,
                            "Невозможно начать работу со звуком.",
                            "Сообщение от controls.exe",
                            MB_ICONINFORMATION );

               /*-------------------------
               Завершаем наше приложение.
               -------------------------*/
               SendMessage ( hWnd, WM_DESTROY, 0 ,0 );
               }
            /*-----------------------------------
            Получаем lpfnAboutDlgPro, указывающее на
            диалоговое окно О программе Controls.
            -----------------------------------*/
            lpfnAboutDlgProc =
            MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);

            /*-----------------------------------------
            Получаем lpfnControlsDlgProc, указывающее на
            диалоговое окно Управление.
            -----------------------------------------*/
            lpfnControlsDlgProc =
            MakeProcInstance (( FARPROC) ControlsDlgProc, ghInst);

            return 0;
```

```
case WM_PAINT:
     hdc = BeginPaint ( hWnd, &ps );
     GetClientRect ( hWnd, &rect );
     DrawText ( hdc,
                "Демонстрация",
                -1,
                &rect,
                DT_SINGLELINE | DT_CENTER | DT_VCENTER );
     EndPaint ( hWnd, &ps );
     return 0;

case WM_COMMAND:
    /*--------------------
    Обработка команд меню.
    ----------------------*/
    switch (wParam)
           {
           case IDM_QUIT:
                /*-----------------------------------
                Пользователь выбрал команду Завершить.
                -------------------------------------*/
                DestroyWindow (hWnd);
                return 0L;

           case IDM_ABOUT :
                /*-----------------------------------
                Пользователь выбрал команду О программе.
                -----------------------------------------*/
                DialogBox ( ghInst,
                            "AboutBox",
                            hWnd,
                            lpfnAboutDlgProc );
                return 0;

           case IDM_CONTROLS:
                /*-----------------------------------
                Пользователь выбрал команду Управление.
                ---------------------------------------*/
                DialogBox ( ghInst,
                            "ControlsBox",
                            hWnd,
                            lpfnControlsDlgProc );
                return 0;

           }/* конец switch (wParam) */


case WM_DESTROY:
     iOpenHelloResult =
     sp_OpenSession ( "c:\\spSDK\\TSfiles\\Hello.ts",
                      SP_NON_STAND_ALONE,
                      0L,
                      SP_TS_TYPE);
```

```
            /*-----------------------------------------
            Ничего не воспроизводим, если сеанс не открыт.
            -----------------------------------------*/
            if ( iOpenHelloResult == SP_NO_ERRORS )
               {
               /*---------------------------------
               Воспроизводим часть звукового файла.
               Сколько играть: от 30000 до 40000
               Фраза: "Good-Bye"
               ---------------------------------*/
               sp_PlayF ( 30000L, 40000L );
               }

            PostQuitMessage (0);
            return 0;

      }/* конец switch (message) */
/*---------------------------
 Сообщение не было обработано.
 ---------------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}/* конец WndProc() */
/*================= конец WndProc() ====================*/

/*======================
 ФУНКЦИЯ: AboutDlgProc()
 ======================*/
/*-------------------------------------------------------------
 ОПИСАНИЕ:
 Это функция обработки диалогового окна О программе Controls.
 -------------------------------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )

{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;

       case WM_COMMAND :
            switch ( wParam )
                   {
                   case IDOK :
                   case IDCANCEL :
                        EndDialog ( hDlg, 0 );
                        return TRUE;
                   }
       }
return FALSE ;
}/* Конец функции. */
/*=============== конец AboutDlgProc() ==============*/
```

```
/*=========================
 ФУНКЦИЯ: ControlsDlgProc()
 =========================*/

/*-------------------------------------------------
 ОПИСАНИЕ:
 Эта функция реализует диалоговое окно Управление.
 -------------------------------------------------*/
BOOL FAR PASCAL _export ControlsDlgProc ( HWND hDlg,
                                          UINT message,
                                          UINT wParam,
                                          LONG lParam )
{
MSG  msg;
HWND hWndCtrl;
HWND hWndCtrlSpeed;
HWND hWndCtrlPosition;

switch ( message )
       {
       case WM_INITDIALOG :
            /*-----------------------------------------
            Получаем образец скорости и отображаем его.
            -----------------------------------------*/
            DisplaySamplingRate ( hDlg);

            /*-----------------------------------------
            Устанавливаем указатель позиции в звуковом
            файле на начало (0).
            -----------------------------------------*/
            glCurrentByte = 0L;


            /*---------------------------------
            Определяем размер звукового файла.
            ---------------------------------*/
            glFileSizeInBytes = sp_GetFileSizeInBytes ();

            /*-----------------------------------------
            Определяем и отображаем длительность звучания
            файла (в секундах).
            -----------------------------------------*/
            DisplayFileSize ( hDlg );

            /*-------------------------------------------------
            Устанавливаем максимальное и минимальное значения
            на линейке изменения скорости воспроизведения.
            -------------------------------------------------*/
            hWndCtrlSpeed = GetDlgItem ( hDlg, SPEED_SB );
            SetScrollRange ( hWndCtrlSpeed,
                             SB_CTL,
                             MIN_SPEED_SCROLL,
                             MAX_SPEED_SCROLL,
                             TRUE );
```

```
            /*-------------------------------------------------
            Устанавливаем максимальное и минимальное значения
            на линейке позиции воспроизведения.
            -------------------------------------------------*/
            hWndCtrlPosition = GetDlgItem ( hDlg, POSITION_SB );
            SetScrollRange ( hWndCtrlPosition,
                             SB_CTL,
                             MIN_POSITION_SCROLL,
                             MAX_POSITION_SCROLL,
                             TRUE );

        /*-------------------------------------------------
        Устанавливаем бегунок на линейке изменения скорости
        воспроизведения в нормальное положение.
        -------------------------------------------------*/
        SetScrollPos ( hWndCtrlSpeed,
                       SB_CTL,
                       NAT_SPEED_SCROLL,
                       TRUE );

        /*---------------------------------------
        Переменная giSpeedScroll изменена.
        Эта переменная отражает позицию бегунка на
        линейке изменения скорости воспроизведения.
        ---------------------------------------*/
        giSpeedScroll = NAT_SPEED_SCROLL ;

        /*-----------------------------------
        Устанавливаем бегунок линейки позиции
        воспроизведения на начало.
        -----------------------------------*/
        SetScrollPos ( hWndCtrlPosition,
                       SB_CTL,
                       MIN_POSITION_SCROLL,
                       TRUE );

        /*-------------------------------------
        Переменная giPositionScroll изменена.
        Эта переменная отражает позицию бегунка
        линейки воспроизведения.
        -------------------------------------*/
        giPositionScroll = MIN_SPEED_SCROLL ;
        return TRUE;
    case WM_HSCROLL:
        /*-----------------------------------------------------
        Пользователь изменил один из параметров воспроизведения.
        -----------------------------------------------------*/
        /*-----------------
        Получаем указатель.
        -----------------*/
        hWndCtrl = HIWORD ( lParam );

        switch ( wParam )
               {
               case SB_PAGEUP:
               case SB_LINEUP:
```

```
            if ( GetDlgItem ( hDlg, SPEED_SB )
                 == hWndCtrl )
               {
               /*-------------------------------------
               Бегунок линейки скорости воспроизведения
               передвинут влево.
               -------------------------------------*/
               giSpeedScroll--;
               ChangeSpeedWasRequested ( hDlg );
               return TRUE;
               }
            else
               {
               /*-------------------------------------
               Бегунок линейки позиции воспроизведения
               передвинут влево.
               -------------------------------------*/
               giPositionScroll-=2;
               ChangePositionWasRequested ( hDlg );
               return TRUE;
               }

         case SB_PAGEDOWN:
         case SB_LINEDOWN:
            if ( GetDlgItem ( hDlg, SPEED_SB )
                 == hWndCtrl )
               {
               /*-------------------------------------
               Бегунок линейке скорости воспроизведения
               передвинут вправо.
               -------------------------------------*/
               giSpeedScroll++;
               ChangeSpeedWasRequested ( hDlg );
               return TRUE;
               }
            else
               {
               /*-------------------------------------
               Бегунок линейки позиции воспроизведения
               передвинут вправо.
               -------------------------------------*/
               giPositionScroll+=2;
               ChangePositionWasRequested ( hDlg );
               return TRUE;
               }

         case SB_THUMBPOSITION:
            if ( GetDlgItem ( hDlg, SPEED_SB )
                 == hWndCtrl )
               {
               /*-------------------------------------
               Бегунок линейки скорости восстановлен
               передвинут в исходное положение.
               -------------------------------------*/
               giSpeedScroll = LOWORD ( lParam );
```

```
                    ChangeSpeedWasRequested ( hDlg );
                    return TRUE;
                    }
                else
                    {
                    /*-----------------------------------
                    Бегунок линейки позиции воспроизведения
                    передвинут в исходное положение.
                    -----------------------------------*/
                    giPositionScroll = LOWORD ( lParam );
                    ChangePositionWasRequested ( hDlg );
                    return TRUE;
                    }

            } /* конец switch(wParam) */

    case WM_COMMAND :
        SetFocus ( hDlg );

        switch ( wParam )
            {
            case FAST_PB:
                /*-----------------------------------
                Пользователь нажал кнопку Быстрее для
                увеличения скорости воспроизведения.
                -----------------------------------*/
                giSpeedScroll++;
                ChangeSpeedWasRequested ( hDlg );
                return TRUE;

            case SLOW_PB:
                /*-----------------------------------
                Пользователь нажал кнопку Медленнее для
                уменьшения скорости воспроизведения.
                -----------------------------------*/
                giSpeedScroll--;
                ChangeSpeedWasRequested ( hDlg );
                return TRUE;

            case PLAY_PB:
                /*-----------------------------------
                Пользователь нажал кнопку Играть для
                начала воспроизведения.
                -----------------------------------*/
                while ( 1 )
                    {
                    glCurrentByte =
                    sp_PlayF ( glCurrentByte,
                               glCurrentByte + 7000L );

                    DisplayLocation ( hDlg, glCurrentByte );

                    if ( glCurrentByte == 0L )
                        break;
```

```
                /*--------------------------------
                Проверка — не нажал ли пользователь
                какую-нибудь клавишу во время
                воспроизведения.
                --------------------------------*/
                if ( PeekMessage ( &msg,
                                   hDlg,
                                   0,
                                   0,
                                   PM_REMOVE ) )
                   {
                   TranslateMessage ( &msg );
                   if ( msg.message == WM_CHAR &&
                        toupper(msg.wParam) == 'S')
                      {
                      /*-----------------------------
                      Пользователь нажал клавишу 'S'
                      во время воспроизведения.
                      -----------------------------*/
                      return TRUE;
                      }
                 if ( msg.message == WM_CHAR &&
                      toupper(msg.wParam) == 204)
                     {
                     /*-----------------------------
                     Если пользователь во время
                     воспроизведения нажал 'М' ('м'),
                     то надо уменьшить скорость.
                     -----------------------------*/
                     giSpeedScroll-=5;
                     ChangeSpeedWasRequested ( hDlg );
                     continue;
                     }
                 if ( msg.message == WM_CHAR &&
                      toupper(msg.wParam) == 193)
                     {
                     /*-----------------------------
                     Если пользователь во время
                     воспроизведения нажал 'Б' ('б'),
                     то надо увеличить скорость.
                     -----------------------------*/
                     giSpeedScroll+=5;
                     ChangeSpeedWasRequested ( hDlg );
                     continue;
                     }

                  }/* конец if(PeekMessage()) */
               }/* конец while (1) */

            return TRUE;

         case EXIT_PB:
         case IDOK :
         case IDCANCEL :
              giSpeedScroll = NAT_SPEED_SCROLL;
```

```
                        sp_SetNewSpeed ( MAX_SPEED_SCROLL,
                                         NAT_SPEED_SCROLL,
                                         giSpeedScroll );

                        EndDialog ( hDlg, 0 );
                        return TRUE;
                    }/* конец switch(wParam) */

        }/* конец switch(wParam) */

return FALSE ;

}/* Конец функции. */
/*================ конец ControlsDlgProc() ==============*/

/*======
 ФУНКЦИЯ
 =======*/
 /*----------------------------------------
 Пользователь изменил позицию бегунка на
 линейке изменения скорости воспроизведения.
 ----------------------------------------*/
void ChangeSpeedWasRequested ( HWND hDlg )
{

HWND  hWndCtrl;

if ( giSpeedScroll   MAX_SPEED_SCROLL )
     giSpeedScroll = MAX_SPEED_SCROLL;

if ( giSpeedScroll   MIN_SPEED_SCROLL )
     giSpeedScroll = MIN_SPEED_SCROLL;

sp_SetNewSpeed ( MAX_SPEED_SCROLL,
                 NAT_SPEED_SCROLL,
                 giSpeedScroll );

hWndCtrl = GetDlgItem ( hDlg, SPEED_SB );
SetScrollPos ( hWndCtrl, SB_CTL, giSpeedScroll, TRUE );

}/* Конец функции. */
/*================== конец функции ==============*/

/*======
 ФУНКЦИЯ
 =======*/
 /*----------------------------------------
 Пользователь изменил позицию бегунка на
 линейке индикации процесса воспроизведения.
 ----------------------------------------*/
void ChangePositionWasRequested ( HWND hDlg )
{

HWND  hWndCtrl;

if ( giPositionScroll >  MAX_POSITION_SCROLL )
     giPositionScroll = MAX_POSITION_SCROLL;
```

```
if ( giPositionScroll <  MIN_POSITION_SCROLL )
     giPositionScroll = MIN_POSITION_SCROLL;

/*----------------------------------------
 Вычислим позицию в звуковом файле, которая
 соответствует положению бегунка линейки.
 ----------------------------------------*/

glCurrentByte = (long)
(( glFileSizeInBytes * giPositionScroll ) / MAX_POSITION_SCROLL );

/*--------------------------------
 Установим бегунок в новую позицию.
 --------------------------------*/

hWndCtrl = GetDlgItem ( hDlg, POSITION_SB );
SetScrollPos ( hWndCtrl,
               SB_CTL,
               giPositionScroll,
               TRUE );

DisplayLocation ( hDlg, glCurrentByte );

}/* Конец функции. */
/*=================== конец функции ==============*/

/*========
 ФУНКЦИЯ
 ========*/
/*-----------------------------------------------
 ОПИСАНИЕ:
 Эта функция изменяет положение бегунка линейки
 позиции воспроизведения и изменяет выводимое
 значение позиции в секундах.
 -----------------------------------------------*/
void DisplayLocation ( HWND hDlg, long lCurrentBytePosition )
{
HWND   hWndCtrl;
float  fCurrentPositionInSec;
char   sPositionInSeconds[15];

hWndCtrl = GetDlgItem ( hDlg, POSITION_SB );

/*----------------------------------------
 Определим текущую позицию в звуковом файле.
 ----------------------------------------*/
fCurrentPositionInSec =
sp_GetCurrentPosInSeconds ( lCurrentBytePosition );

/*----------------------------
 Отобразим позицию в секундах.
 ----------------------------*/
sprintf ( sPositionInSeconds,
          "%.2f",
          fCurrentPositionInSec );
SetDlgItemText (hDlg, POSITION_TEXT, sPositionInSeconds );
```

```
/*-----------------------------------------
 Вычислим новое положение бегунка на линейке.
 -----------------------------------------*/
giPositionScroll =
(int)(( MAX_POSITION_SCROLL * lCurrentBytePosition )/
        glFileSizeInBytes);

/*-----------------------------------
 Установим бегунок в новое положение.
 -----------------------------------*/
SetScrollPos ( hWndCtrl,
               SB_CTL,
               giPositionScroll,
               TRUE );

}/* Конец функции. */
/*================ конец функции ==================*/


/*=======
 ФУНКЦИЯ
 ========*/
/*=============================================================
 ОПИСАНИЕ:
 Эта функция отображает длительность звучания файла в секундах.
 =============================================================*/
void DisplayFileSize ( HWND hDlg)
{
char sLengthInSeconds[15];

sprintf ( sLengthInSeconds,
          "%.2f",
          sp_GetFileSizeInSeconds() );

SetDlgItemText ( hDlg,
                 TOTAL_TEXT,
                 sLengthInSeconds );

}/* Конец функции. */
/*============ конец функции ========*/


/*=======
 ФУНКЦИЯ
 ========*/
/*=================================================
 ОПИСАНИЕ:
 Эта функция отображает частоту дискретизации файла.
 =================================================*/
void DisplaySamplingRate ( HWND hDlg)
{
char sSamplingRate[15];

sprintf ( sSamplingRate,
          "%.2f",
          (float)sp_GetSamplingRate() );
```

```
SetDlgItemText ( hDlg,
                 SAMPLING_TEXT,
                 sSamplingRate );

}/* Конец функции. */
/*============= конец функции ========*/
```

Листинг 8.8. Controls.rc

```
/*=================================================
 ИМЯ ФАЙЛА: controls.rc
 ---------
 ОПИСАНИЕ ФАЙЛА:
 ---------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 =================================================*/
/*--------
 #include
 --------*/
#include <windows.h>
#include "Controls.h"

/*---
 Меню
 ----*/
Controls  MENU
BEGIN
   POPUP  "&Меню"
      BEGIN
          MENUITEM "&Завершить", IDM_QUIT
          MENUITEM "&О программе", IDM_ABOUT
          MENUITEM "&Управление", IDM_CONTROLS
      END
END

/*---------------------------------------
 Определение значка магнитофонной кассеты.
 Имя файла: Tape.ico
 Имя значка: IconOfTape
 ---------------------------------------*/
IconOfTape ICON Tape.ico

/*---------------------------------
 Диалоговое окно О программе Controls.
 ---------------------------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "О программе Controls"
/* необходим локализованный  шрифт MS Sans Serif */
FONT 8, "MS Sans Serif"
```

```
BEGIN
    PUSHBUTTON      "OK", IDOK, 64, 75, 40, 14
    CTEXT           "(C) Copyright Gurewich 1992, 1993",
                    -1, 13, 47, 137, 18
    ICON            "IconOfTape", -1, 14, 12, 18, 20
END

/*-------------------------
 Диалоговое окно Управление.
 -------------------------*/
ControlsBox DIALOG 32, 34, 280, 178
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Управление"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "В&ыход", EXIT_PB, 118, 159, 40, 14
    GROUPBOX        "Частота дискрет. в Гц", -1,
                    11, 10, 89, 40
    GROUPBOX        "Позиция в секундах", -1
                    8, 58, 90, 40
    SCROLLBAR       POSITION_SB, 7, 110, 267, 6
    SCROLLBAR       SPEED_SB, 109, 11, 165, 21
    PUSHBUTTON      "Нажмите 'S' для останова", -1,
                    182, 159, 85, 14
    PUSHBUTTON      "&Играть", PLAY_PB, 114, 121, 159, 31
    CTEXT           "0 Sec", POSITION_TEXT, 29, 76, 50, 8
    ICON            "IconOfTape", -1, 5, 132, 18, 20
    ICON            "IconOfTape", -1, 22, 151, 18, 20
    ICON            "IconOfTape", -1, 54, 127, 18, 20
    ICON            "IconOfTape", -1, 71, 152, 18, 20
    PUSHBUTTON      "&Медленнее", SLOW_PB, 108, 34, 27, 12
    PUSHBUTTON      "&Быстрее", FAST_PB, 251, 34, 25, 12
    LTEXT           "Обычно", -1, 179, 36, 28, 8
    LTEXT           "назад", -1, 7, 100, 34, 8
    LTEXT           "вперед", -1, 247, 100, 32, 8
    CTEXT           "0    ", SAMPLING_TEXT, 33, 28, 45, 8
    LTEXT           "Неизвестно", TOTAL_TEXT, 171, 75, 32, 8
    GROUPBOX        "Общая длина в секундах", -1, 109, 58, 154, 40
END
```

Листинг 8.9. Controls.def

```
;==================================
; Файл описаний для Controls.c
;==================================

NAME        Controls

DESCRIPTION  'Программа Controls. (C) Copyright Gurewich 1992, 1993'

EXETYPE     WINDOWS

STUB        'WINSTUB.EXE'
```

```
CODE  PRELOAD MOVEABLE DISCARDABLE

DATA  PRELOAD MOVEABLE MULTIPLE

HEAPSIZE    1024
STACKSIZE   8192
```

Листинг 8.10. Controls.mak

```
#=============
# Controls.mak
#=============

Controls.exe : Controls.obj Controls.h Controls.def Controls.res
    link /nod Controls.obj, Controls.exe, NUL, \
       slibcew.lib oldnames.lib libw.lib commdlg \
       c:\spSDK\TegoWlib\TegoWin.lib, \
       Controls.def
    rc -t Controls.res

Controls.obj : Controls.c Controls.h
   cl -c -G2sw -Ow -W3 -Zp Controls.c

Controls.res : Controls.rc Controls.h Tape.ico
   rc -r Controls.rc
```

Новые sp_-функции в программе Controls

В программе Controls мы применили несколько новых sp_-функций. Ниже эти функции описаны более подробно.

Функция sp_GetFileSizeInBytes()

Размер звукового файла может быть получен из с помощью вызова функции sp_GetFileSizeInBytes ():

```
glFileSizeInBytes = sp_GetFileSizeInBytes ();
```

Прототип функции GetFileSizeInBytes() объявлен в файле c:\spSDK\TegoWlib\sp4Win.h:

```
long sp_GetFileSizeInBytes ( void );
```

Функции не передаются никакие параметры; возвращает она длинное целое, которое означает размер открытого звукового файла в байтах.

Примечание:

Функции sp_GetFileSizeInBytes () параметры не передаются.

Возвращаемое значение: Длинное целое, которое означает длину в байтах открытого звукового файла.

Перед вызовом функции sp_GetFileSizeInBytes() уже должна быть вызвана функция sp_OpenSession().

Функция sp_CurrentPosInSeconds()

Текущая позиция в открытом звуковом файле может быть получена с помощью вызова функции sp_CurrentPosInSeconds().

Прототип функции sp_CurrentPosInSeconds() описан в файле c:\spSDK\TegoWlib\sp4Win.h следующим образом:

```
float sp_CurrentPosInSeconds ( long );
```

Функции передается один параметр — длинное целое, которое указывает текущую позицию воспроизведения в байтах.

Функция возвращает вещественное число, которое представляет собой выраженное в секундах значение текущей позиции в звуковом файле.

Примечание:

Функции sp_CurrentPosInSeconds() передается один параметр.

Параметр: Длинное целое. Значение параметра представляет собой выраженную в байтах текущую позицию в звуковом файле при его воспроизведении.

Возвращаемое значение: Вещественное число. Представляет собой выраженное в секундах значение текущей позиции в звуковом файле при его воспроизведении.

Функция sp_GetSamplingRate()

Частота дискретизации открытого звукового файла может быть получена с помощью вызова функции sp_GetSamplingRate().

Прототип функции sp_GetSamplingRate() описан в файле c:\spSDK\TegoWlib\sp4Win.h следующим образом:

```
long sp_GetSamplingRate ( void );
```

Функции не передается ни один параметр; возвращает она длинное целое, которое представляет собой значение частоты дискретизации открытого звукового файла

Примечание:

Функции sp_GetSamplingRate() параметры не передаются.

Возвращаемое значение: Длинное целое, которое представляет собой значение частоты дискретизации открытого звукового файла. Перед вызовом функции sp_GetSampling-Rate() уже должна быть вызвана функция sp_OpenSession().

О программе Controls

Программа Controls очень похожа на программу Rotate, за исключением двух главных отличий:

1. В программе Controls больше элементов управления (командных кнопок, линеек прокрутки и т.п.).
2. Программа Controls позволяет пользователю останавливать воспроизведение. Кроме того, во время воспроизведения возможно изменение его скорости (дополнительная информация содержится в одном из следующих параграфов).

Отображение позиции в звуковом файле во время воспроизведения

В варианте PLAY_PB мы выполняем цикл while(1), в котором реализовано воспроизведение звукового файла фрагментами по 7000 байт каждый:

```
case PLAY_PB:
   /*-----------------------------------
   Пользователь нажал кнопку Играть для
   начала воспроизведения.
   -----------------------------------*/
   while ( 1 )
         {
         glCurrentByte =
         sp_PlayF ( glCurrentByte,
                    glCurrentByte + 7000L );

         DisplayLocation ( hDlg, glCurrentByte );

         if ( glCurrentByte == 0L )
              break;
        /*---------------------------------
        Проверка — не нажал ли пользователь
        во время воспроизведения
        какую-нибудь клавишу.
        ---------------------------------*/

  }/* конец while (1) */
  return TRUE;
```

После воспроизведения 7000 байт вызывается функция DispiayLocation(). Эта функция передвигает бегунок линейки позиции воспроизведения и отображает выраженное в секундах значение текущей позиции в файле.

Для получения этого значения в теле функции DisplayLocation() мы используем функцию sp_GetCurrent PosInSeconds():

```
fCurrentPositionInSec =
sp_GetCurrentPosInSeconds ( lCurrentBytePosition );
```

Затем отображаем на экране следующее значение:

```
sprintf ( sPositionInSeconds,
          "%.2f",
          fCurrentPositionInSec );

SetDlgItemText ( hDlg,
                 POSITION_TEXT,
                 sPositionInSeconds );
```

Функция DisplayLocation() передвигает бегунок линейки позиции воспроизведения в новое положение, которое вычисляется исходя из нового значения переменной giPositionScroll. Затем мы вызываем функцию SetScrollPos():

```
giPositionScroll =
(int)(( MAX_POSITION_SCROLL * lCurrentBytePosition )/
        glFileSizeInBytes);

SetScrollPos ( hWndCtrl,
               SB_CTL,
               giPositionScroll,
               TRUE );
```

Остановка воспроизведения по инициативе пользователя

Очень важно понять, что воспроизведение не может быть остановлено во время воспроизведения фрагмента.

Если пользователь нажимает клавишу [S] во время выполнения цикла while(1), то Windows генерирует соответствующее сообщение, которое обрабатывается программой. После окончания воспроизведения фрагмента наша программа проверяет, не поступило ли для нее сообщение:

```
if ( PeekMessage ( &msg,
                   hDlg,
                   0,
                   0,
                   PM_REMOVE ) )
   {
   TranslateMessage ( &msg );
   ..............................
   .... Анализируем сообщение ....
   ..............................
   }
```

Если для нашей программы сообщений нет, то вновь выполняем итерацию цикла while(1) и воспроизводим еще один фрагмент.

Если для нашей программы пришло сообщение, то мы анализируем его последовательностью операторов if (хотя, в общем-то, могли бы воспользоваться и оператором switch).

В следующем операторе if рассматривается вариант нажатия клавиши[S] ([s]):

```
if ( msg.message == WM_CHAR &&
     toupper(msg.wParam) == 'S')
   {
   /*-----------------------------------
   Пользователь для останова нажал
   во время воспроизведения клавишу 'S'.
   -----------------------------------*/
   return TRUE;
   }
```

Возможность изменения скорости во время воспроизведения

Аналогичным образом отслеживаем варианты, в которых пользователь нажимает клавишу [М] для уменьшения скорости воспроизведения или [Б] — для ее увеличения:

```
if ( msg.message == WM_CHAR &&
     toupper(msg.wParam) == 204 )
   {
   /*-------------------------------
   Если пользователь во время
   воспроизведения нажал 'М' ('м'),
   то надо уменьшить скорость
   воспроизведения.
   -------------------------------*/
   giSpeedScroll-=5;
   ChangeSpeedWasRequested ( hDlg );
   continue;
   }

if ( msg.message == WM_CHAR &&
     toupper(msg.wParam) == 193)
```

```
{
/*------------------------------
Если пользователь во время
воспроизведения нажал 'B' ('б'),
то надо увеличить скорость
воспроизведения.
------------------------------*/
giSpeedScroll+=5;
ChangeSpeedWasRequested( hDlg );
continue;
}
```

Изменение размера фрагмента при воспроизведении

Размер фрагмента для воспроизведения определяет скорость, с которой программа реагирует на нажатие клавиш во время воспроизведения. В нашей программе мы воспроизводим файл фрагментами по 7000 байт.

Для файла типа TS используем следующую формулу:

$$\text{Время в секундах} = \frac{\text{Количество байтов}}{\text{Частота дискретизации}}$$

Звуковой файл был записан с частотой дискретизации, равной 8000 Гц, следовательно, для воспроизведения 7000 байт понадобится 0,875 с:

$$\text{Время} = \frac{7000}{8000} = 0{,}875 \text{ с}$$

Это означает, что если пользователь нажал клавишу [S] в самом начале воспроизведения фрагмента, то от момента нажатия клавиши до прекращения воспроизведения пройдет не более 0,875 с.

Глава 9. Программы на базе Generic2: многозадачность

До сих пор мы создавали приложения Windows на основе программы Generic1. Но как вы, вероятно, заметили, такие программы не используют мощных многозадачных возможностей оболочки Windows. Фактически техника, которая в них применяется, не отличается от техники, используемой программами на языке C для DOS.

Программы же, основанные на Generic2, напротив, используют все преимущества многозадачных возможностей оболочки Windows.

Работа оболочки Windows

Оболочка Windows позволяет практически одновременно выполнять несколько приложений. Рассмотрим, как это делается.

Цикл обработки сообщений программы Generic1

В данном параграфе мы рассмотрим цикл обработки сообщений программы Generic1.

Начнем с функции WinMain(), которую должно содержать каждое Windows- приложение. Предположим, что в нашем сеансе Windows открыты три программы — App1, App2, и App3. Предположим также, что эти три программы следующим образом используют в своих циклах сообщений функцию GetMessage():

```
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage  ( &msg );
      }
```

Функция GetMessage() программы App1 передает управление оболочке Windows. Оболочка Windows возвращает управление App1 **ТОЛЬКО** тогда, когда существует сообщение для данной программы (например, пользователь щелкнул мышью внутри окна App1). Если такое сообщение есть, то управление передается App1 и начинает выполняться команда, следующая за вызовом функции GetMessage(). Обычно такой командой является вызов функции TranslateMessage() или DispatchMessage(). Функция TranslateMessage() транслирует сообщение, а функция DispatchMessage() — возвращает его оболочке Windows. Оболочка Windows выполняет функцию WndProc() программы App1. После того как программа завершит выполнение функции WndProc(), управление возвращается оболочке Windows, которая вновь передает его оператору, следующему за вызовом функции DispatchMessage(). В нашем цикле обработки сообщений таким оператором является начало цикла while() — вызов функции GetMessage(). Если сообщений для программы App1 нет, то Windows передает управление не App1, а циклу обработки сообщений программы App2 (или App3), если для них имеются какие-либо сообщения. В следующий раз управление вернется программе App1 только тогда, когда для нее будет сообщение.

Важно понимать, что если нет сообщений ни для программы App1, ни для App2, ни для App3, то управление остается у оболочки Windows до тех пор, пока не появится сообщение для какой-нибудь из этих программ. И только после этого Windows передаст управление той программе, которой это сообщение адресовано.

Цикл while() завершится, если функция GetMessage() возвратит FALSE, а это произойдет в том случае, когда поступит сообщение WM_QUIT. Поступление данного сообщения вызывает прерывание цикла while() и завершение программы.

Цикл обработки сообщений программы Generic2

Теперь предположим, что цикл обработки сообщений программы App1 модифицирован следующим образом:

```
while (TRUE)
      {
      if ( PeekMessage ( &msg, NULL, 0, 0, PM_REMOVE ) )
         {
         /*--------------------------------------
         Существуют сообщения в очереди сообщений.
         Проверим, что — это сообщение WM_QUIT.
         --------------------------------------*/
         if ( msg.message == WM_QUIT )
            {
            /*------------------------
            Сообщение WM_QUIT принято.
            ------------------------*/
            break; /* Завершение приложения */
         /*--------------------------------------
         Сообщение принято.
         Принятое сообщение — не WM_QUIT.
         --------------------------------------*/
          TranslateMessage ( &msg );
          DispatchMessage  ( &msg );
          }
       else
          {
          ..........................................
          ...... Вариант отсутствия сообщений ........
          ..........................................
          }
      } /* Конец while (TRUE) */
```

Это цикл обработки сообщений, который используется программами типа Generic2. Он похож на цикл обработки сообщений программы Generic1, однако вместо функции GetMessage() в него включена функция PeekMessage().

Когда в программе App1 вызывается функция PeekMessage(), управление передается оболочке Windows. Оболочка Windows проверяет наличие любых сообщений для программ App2 и App3 и, если таковые имеются, передает управление циклу обработки сообщений App2 или App3.

Между функциями GetMessage() и PeekMessage() имеются два существенных различия.

Когда вызывается функция GetMessage() программы App1, оператор, следующий за вызовом этой функции, будет выполнен *только в том случае, когда существует сообщение* для данной программы. Когда применяется функция PeekMessage(), оболочка Windows возвращает управление оператору, который следует за вызовом PeekMessage(), *в любом случае* — существует сообщение для данной программы или нет. Таким образом, оператор, следующий за вызовом функции PeekMessage(), выполняется *всегда.* Возвращаемое функцией PeekMessage() значение указывает, существует для данной программы какое-нибудь сообщение или нет. Цикл обработки сообщений программы App1 проверяет это возвращаемое значение, и если сообщение есть, то выполняется проверка, не является ли это сообщение WM_QUIT. Если поступило сообщение WM_QUIT, то цикл while(TRUE) прерывается и программа завершается. Если поступившее сообщение —

не WM_QUIT, то выполняются функции TranslateMessage() и DispatchMessage(). Функция TranslateMessage() транслирует сообщение, а функция DispatchMessage() — возвращает его оболочке Windows. Затем Windows выполняет функцию WndProc() программы App1. Когда выполнение WndProc() завершено, управление вновь возвращается оболочке Windows и Windows возвращает его оператору, следующему за вызовом функции DispatchMessage(). В нашем случае таким оператором является начало цикла while(TRUE). Это означает, что следующим оператором, который выполнится, будет вызов функции PeekMessage() и, следовательно, весь процесс повторится.

Поскольку мы применили здесь функцию PeekMessage(), то оболочка Windows передаст управление нашей программе в любом случае — независимо от того, существует для нее какое-либо сообщение или нет. Поэтому нам необходимо дополнить текст программы фрагментом для обработки варианта отсутствия сообщений. Этот фрагмент помещается после ключевого слова else оператора if(PeekMessage()).

Программа Generic2

В цикле обработки сообщений программы Generic2 используется функция PeekMessage(). Главное окно программы Generic2 приведено на рис. 9.1.

Меню программы Generic2 (см. рис. 9.2) содержит шесть команд — **Завершить**, **О программе**, **Разрешить фоновое воспроизведение**, **Запретить фоновое воспроизведение**, **Запретить мышь при воспроизведении** и **Разрешить мышь при воспроизведении**. Диалоговое окно **О программе Generic2** приведено на рис. 9.3.

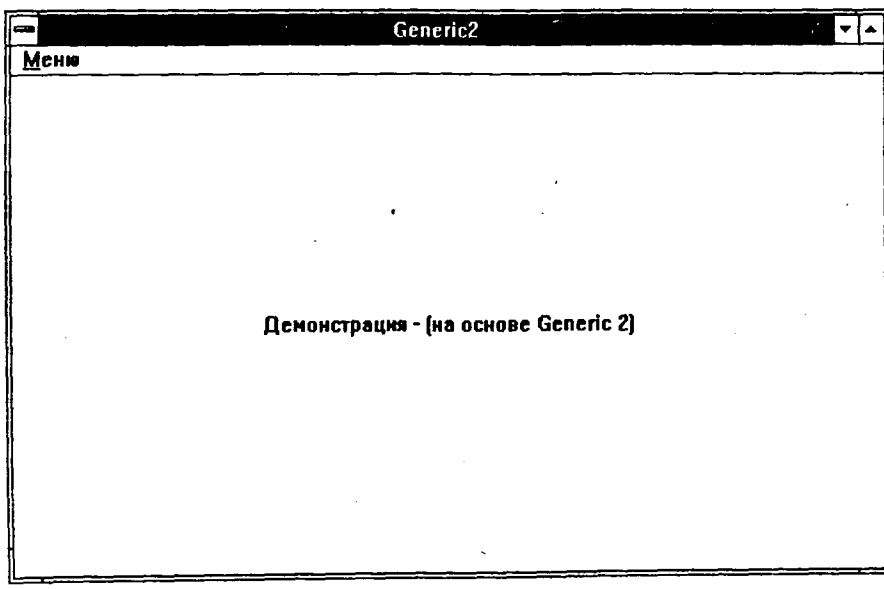


Рис. 9.1. Главное окно программы Generic2

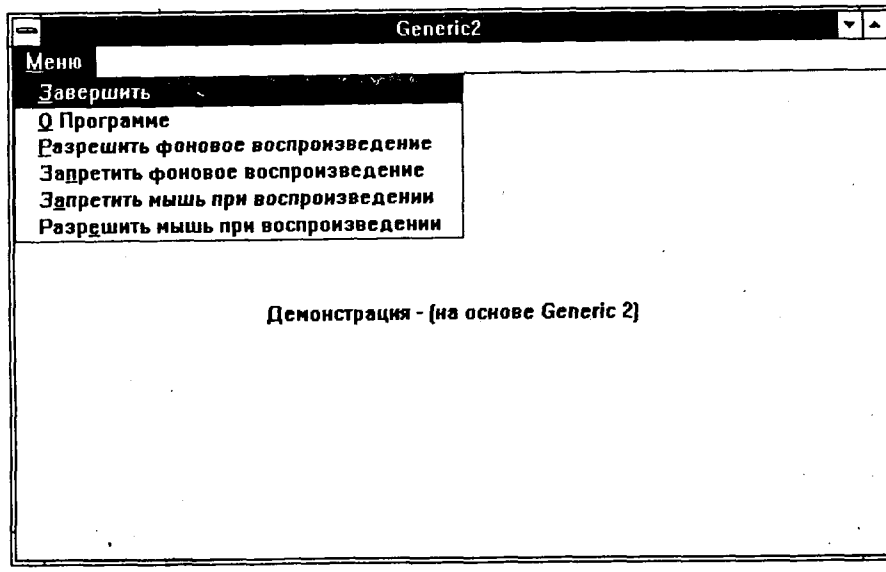


Рис. 9.2. Меню программы Generic2

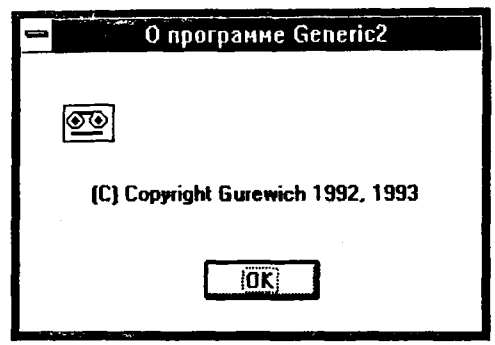


Рис. 9.3. Диалоговое окно О программе Generic2

Компиляция и компоновка программы Generic2

Для компиляции и компоновки программы Generic2 компилятором фирмы Microsoft необходимо выполнить следующие действия:

- Убедиться в том, что компьютер находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4Win.
- После приглашения DOS ввести:

```
NMAKE Generic2.mak    [Enter]
```

Для компиляции и компоновки программы Generic2 компилятором фирмы Borland выполните такие действия:

- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
MAKE -f Generic2.bmk    [Enter]
```

Выполнение программы Generic2

Для выполнения программы Generic2 необходимо:

- Выбрать из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- С помощью диалогового окна **Пролистать** выбрать файл

```
c:\spSDK\samp4Win\Generic2.exe
```

Появляется главное окно программы, и в фоновом режиме воспроизводится звуковой файл.

Для запрета фонового воспроизведения нажмите клавиши [Alt+M], а затем с помощью клавиш со стрелками выберите из меню программы команду **Запретить фоновое воспроизведение**.

Для разрешения фонового воспроизведения нажмите клавиши [Alt+M] и с помощью клавиш со стрелками выберите из меню программы команду **Разрешить фоновое воспроизведение**.

Использование мыши при воспроизведении может быть разрешено или запрещено нажатием клавиш [Alt+M] с последующим выбором из меню программы команд **Разрешить мышь при воспроизведении** или **Запретить мышь при воспроизведении**.

После запуска данной программы можно переключиться в Диспетчер Программ Windows и вызвать, например, Paintbrush. При работе с программой Paintbrush вы будете слышать воспроизведение звукового файла. Кроме того, можно выбрать в окне группы программ Главная пиктограмму **MS-DOS** и войти в MS-DOS. В зависимости от особенностей вашего компьютера операция запуска MS-DOS может занять несколько секунд. В конце концов ПК перейдет в режим MS-DOS, а вы будете слышать воспроизведение звукового файла. Даже после запуска DOS-программы воспроизведение не прекратится.

Для возврата в Windows после приглашения DOS введите **Exit**.

Файлы программы Generic2

Программа Generic2 состоит из следующих файлов:

Generic2.h	#include-файл программы
Generic2.c	Текст программы на C
Generic2.rc	Файл ресурсов программы
Generic2.def	Файл определения модуля программы
Generic2.mak	Make-файл программы
Tape.ico	Пиктограмма программы

Эти файлы записаны на вашем винчестере в директории c:\spSDK\Samp4Win\. Все файлы Generic2 приведены в листингах 9.1–9.5.

Листинг 9.1. Generic2.h

```
/*---------
 прототипы
 ----------*/
long FAR PASCAL _export WndProc          ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export AboutDlgProc     ( HWND, UINT, UINT, LONG ) ;
```

```
/*------
 #define
 -------*/
#define IDM_QUIT          1 /* Команда меню Завершить    */
#define IDM_ABOUT         2 /* Команда меню О программе */
#define IDM_ENABLE_PLAY   3 /* Команда меню Разрешить воспроизведение */
#define IDM_DISABLE_PLAY  4 /* Команда меню Запретить воспроизведение */
#define IDM_ENABLE_MOUSE  5 /* Команда меню Разрешить мышь */
#define IDM_DISABLE_MOUSE 6 /* Команда меню Запретить мышь */

/*--------------------
 Глобальные переменные
 ---------------------*/
char       gszAppName[] = "Generic2" ; /* Имя нашего приложения. */
HINSTANCE  ghInst;                     /* текущий экземпляр.     */
int        giPlayEnable;
long       glCurrentBackGndByte;
```

Листинг 9.2. Generic2.c

```
/*=================================================================
 ПРОГРАММА: Generic2.c
 ----------
 (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

 НАЗНАЧЕНИЕ ПРОГРАММЫ:
 ---------------------
 Это программа типа Generic2.

 Эта программа воспроизводит музыку в фоновом режиме
 (в бесконечном цикле).

 Пользователь может выполнять любые приложения Windows,
 так же, как и приложения DOS, а ПК будет проигрывать
 звуки в фоновом режиме.

 =================================================================*/
/*--------
 #include
 --------*/
/*-----------------------------------------------
 windows.h требуется во всех приложениях Windows.
 -----------------------------------------------*/
#include <windows.h>
/*-------------------------------------------------
 sp4Win.h требуется для того, чтобы в этих приложениях
 могли быть использованы функции sp_, а также макросы
 SP_ и #define из библиотеки звуковой поддержки.
 -------------------------------------------------*/
 #include "c:\spSDK\TegoWlib\sp4Win.h"

/*-----------------------------------------------------------
 Определения и прототипы, характерные для этого приложения.
 -----------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\Generic2.h"
```

```
/*-------------------------------------------------------
 Для функций C, требующих стандартные #include-файлы C.
 -------------------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>

/*=================
 ФУНКЦИЯ: WinMain()
 =================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HWND       hWnd;   /* Идентификатор окна нашего приложения.    */
MSG        msg;    /* Сообщение, которое будет обрабатываться */
                   /* нашим приложением.                       */
WNDCLASS   wc;     /* Класс окна нашего приложения.            */

int        iOpenResult; /* Результат открытия звукового сеанса. */

/*---------------------------------------
 Создадим глобальную переменную hInstance.
 ---------------------------------------*/
ghInst = hInstance;

/*-----------------------------------------------------------
 Обновляем структуру класса окна и регистрируем класс окна.
 -----------------------------------------------------------*/
if ( !hPrevInstance )
   {
   /*---------------------------------------
   Если условие if удовлетворено, это самый
   первый запуск приложения.
   ---------------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfTape" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW      ) ;
   wc.hbrBackground = GetStockObject ( WHITE_BRUSH ) ;
   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = gszAppName ;

  /*----------------
  Регистрируем окно.
  ----------------*/
  RegisterClass ( &wc );

  }/* конец условия if( !hPrevInstance ) */
```

```
/*-------------------------
 Открываем звуковой сеанс.
 -------------------------*/
iOpenResult =
sp_OpenSession ( "c:\\spSDK\\Sfiles\\Day.s",
                 SP_NON_STAND_ALONE,
                 0L,
                 SP_S_TYPE );
/*-------------------------------------------
 Завершаем работу, если невозможно открыть файл.
 -------------------------------------------*/
if ( iOpenResult != SP_NO_ERRORS )
   {
   MessageBox ( NULL,
                "Невозможно открыть звуковой сеанс!",
                "Сообщение от Generic2.exe",
                MB_ICONINFORMATION );
   /*--------------------
   Завершаем приложение.
   --------------------*/
   SendMessage ( hWnd,
                 WM_DESTROY,
                 0,
                 0 );

   }/* конец условия if(iOpenResult!=SP_NO_ERRORS) */

/*---------------------------------------
 Разрешаем работу мыши при воспроизведении.
 ---------------------------------------*/
sp_EnableMouseDuringPlay ();

/*-----------------------------
 Создаем окно нашего приложения.
 -----------------------------*/
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );

/*-------------------------------------------
 Показываем и обновляем окно нашего приложения.
 -------------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );
/*---------------------------------------
 Устанавливаем переменную giPlayEnable таким
 образом, чтобы разрешить воспроизведение.
 ---------------------------------------*/
giPlayEnable = 1;
```

```
/*-----------------------------------------
 Устанавливаем счетчик воспроизведенного байта
 в начало звукового файла.
 -----------------------------------------*/
glCurrentBackGndByte = 0L;
/*-----------------------------------
 Цикл обработки сообщений (типа Generic2).
 -----------------------------------*/
while (TRUE)
      {
      if ( PeekMessage ( &msg, NULL, 0, 0, PM_REMOVE ) )
         {
         /*-----------------------------------
         В очереди сообщений существует сообщение.
         Проверим, что это сообщение — WM_QUIT.
         -----------------------------------*/
         if ( msg.message == WM_QUIT )
            {
            /*-------------------------
            Принято сообщение WM_QUIT.
            -------------------------*/
            break; /* Завершим приложение. */
            }
         /*-------------------------------
         Сообщение принято.
         Принятое сообщение — не WM_QUIT.
         -------------------------------*/
         TranslateMessage ( &msg );
         DispatchMessage  ( &msg );
         }

      else
         {
         /*---------------------------------
         В очереди сообщений сообщений нет.
         ---------------------------------*/
         if ( giPlayEnable == 1 )
            {
            /*-------------------
            Можно воспроизводить.
            ---------------------*/
            glCurrentBackGndByte =
            sp_PlayF ( glCurrentBackGndByte,
                       glCurrentBackGndByte + 2000L );
            }
         }
     }/* конец цикла while (TRUE) */

return msg.wParam ;

} /* Конец функции. */
/*========================= конец WinMain() ====================*/
```

```
/*=================
 ФУНКЦИЯ: WndProc()
 =================*/
/*-------------------------------
 НАЗНАЧЕНИЕ: Обработка сообщений.
 -------------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HDC            hdc;    /* Необходима для отображения текста. */
PAINTSTRUCT     ps;    /* Необходима для отображения текста. */
RECT          rect;    /* Необходима для отображения текста. */

static FARPROC lpfnAboutDlgProc ; /* Для диалогового окна О Программе.*/

switch ( message )
       {
       case WM_CREATE:
            /*-----------------------------------------------------
            Получаем lpfnAboutDlgProc диалогового окна О Программе.
            -----------------------------------------------------*/
            lpfnAboutDlgProc =
            MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);

            return 0;

       case WM_PAINT:
            hdc = BeginPaint ( hWnd, &ps );
            GetClientRect ( hWnd, &rect );
            DrawText ( hdc,
                       "Демонстрация — (на основе Generic 2)",
                       -1,
                       &rect,
                       DT_SINGLELINE | DT_CENTER | DT_VCENTER );
            EndPaint ( hWnd, &ps );
            return 0;

       case WM_COMMAND:
            SetFocus ( hWnd );
            /*-------------------------------------
            Пользователь выбрал команду из меню или
            нажал командную кнопку.
            -------------------------------------*/
            switch (wParam)
                   {
                   case IDM_ENABLE_PLAY:
                        giPlayEnable = 1 ;
                        return 0L;

                   case IDM_DISABLE_PLAY:
                        giPlayEnable = 0 ;
                        return 0L;
```

```
                case IDM_ENABLE_MOUSE:
                     /*-------------------------------------
                     Разрешим работу мыши при воспроизведении.
                     -------------------------------------*/
                     sp_EnableMouseDuringPlay ();
                     return 0L;

                case IDM_DISABLE_MOUSE:
                     /*-------------------------------------
                     Запретим работу мыши при воспроизведении.
                     -------------------------------------*/
                     sp_DisableMouseDuringPlay ();
                     return 0L;

                case IDM_QUIT:
                     /*-----------------------------------
                     Пользователь выбрал команду Завершить.
                     -----------------------------------*/
                     DestroyWindow (hWnd);
                     return 0L;

                case IDM_ABOUT :
                     /*-----------------------------------
                     Пользователь выбрал команду О Программе.
                     -----------------------------------*/
                     DialogBox ( ghInst,
                                 "AboutBox",
                                 hWnd,
                                 lpfnAboutDlgProc );
                     return 0;


           }/* конец оператора switch(wParam) в варианте WM_COMMAND */

      case WM_DESTROY:
           PostQuitMessage (0);
           return 0;
      }/* конец оператора switch (message) */

/*-----------------------
 Сообщение не обработано.
 -----------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}
/*================== конец WndProc() ====================*/


/*======================
 ФУНКЦИЯ: AboutDlgProc()
 ======================*/
/*--------------------------------------
 НАЗНАЧЕНИЕ:
 Это процедура диалогового окна О Программе.
 --------------------------------------*/
```

```
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;
       case WM_COMMAND :
            switch ( wParam )
                   {
                   case IDOK :
                   case IDCANCEL :
                        EndDialog ( hDlg, 0 );
                        return TRUE;
                   }
       }
return FALSE ;
}/* Конец функции. */
/*================ Конец AboutDlgProc() ==============*/
```

Листинг 9.3. Generic2.rc

```
/*================================================
 ИМЯ ФАЙЛА: Generic2.rc
 ----------

 НАЗНАЧЕНИЕ ПРОГРАММЫ:
 ---------------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 ================================================*/
/*-------
 #include
 --------*/
#include <windows.h>
#include "Generic2.h"

/*---
 Меню
 ----*/
Generic2  MENU
BEGIN
  POPUP  "&Меню"
    BEGIN
     MENUITEM "&Завершить",  IDM_QUIT
     MENUITEM "&О Программе", IDM_ABOUT
     MENUITEM "&Разрешить фоновое воспроизведение",  IDM_ENABLE_PLAY
     MENUITEM "За&претить фоновое воспроизведение",  IDM_DISABLE_PLAY
     MENUITEM "З&апретить мышь при воспроизведении", IDM_DISABLE_MOUSE
     MENUITEM "Разр&ешить мышь при воспроизведении", IDM_ENABLE_MOUSE
    END
END
```

```
/*------------------------------
 Определение пиктограммы кассеты.
 Имя файла: Tape.ico
 Имя пиктограммы: IconOfTape
 ------------------------------*/
IconOfTape ICON Tape.ico

/*---------------------------
 Диалоговое окно О Программе.
 ---------------------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "О Программе Generic2"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON    "OK", IDOK, 64, 75, 40, 14
    CTEXT         "(C) Copyright Gurewich 1992, 1993", -1,
                  13, 47, 137, 18
    ICON          "IconOfTape", -1, 14, 12, 18, 20
END
```

Листинг 9.4. Generic2.def

```
;=======================================
; Файл определения модуля для Generic2.c
;=======================================

NAME        Generic2

DESCRIPTION  'Программа Generic2. (C) Copyright Gurewich 1992, 1993'

EXETYPE     WINDOWS

STUB        'WINSTUB.EXE'

CODE  PRELOAD MOVEABLE DISCARDABLE

DATA  PRELOAD MOVEABLE MULTIPLE

HEAPSIZE    1024
STACKSIZE   8192
```

Листинг 9.5. Generic2.mak

```
#=============
# Generic2.mak
#=============

Generic2.exe : Generic2.obj Generic2.h Generic2.def Generic2.res
    link /nod Generic2.obj, Generic2.exe, NUL, \
       slibcew.lib oldnames.lib libw.lib commdlg \
       c:\spSDK\TegoWlib\TegoWin.lib, \
       Generic2.def
    rc -t Generic2.res
```

```
Generic2.obj : Generic2.c Generic2.h
   cl -c -G2sw -Ow -W3 -Zp Generic2.c

Generic2.res : Generic2.rc Generic2.h Tape.ico
   rc -r Generic2.rc
```

Функция WinMain() Generic2

Функция WinMain() Generic2 открывает звуковой сеанс как неавтономный сеанс со звуковым файлом типа S c:\spSDK\Sfiles\Day.s. Если это невозможно, программа завершается:

```
/*------------------------
 Открываем звуковой сеанс.
 ------------------------*/
iOpenResult =
sp_OpenSession ( "c:\\spSDK\\Sfiles\\Day.s",
                 SP_NON_STAND_ALONE,
                 0L,
                 SP_S_TYPE );
/*-------------------------------------------
 Завершаем работу, если невозможно открыть файл.
 -------------------------------------------*/
if ( iOpenResult != SP_NO_ERRORS )
   {
   MessageBox ( NULL,
                "Невозможно открыть звуковой сеанс!",
                "Сообщение от Generic2.exe",
                MB_ICONINFORMATION );

   /*--------------------
   Завершаем приложение.
   --------------------*/
   SendMessage ( hWnd,
                 WM_DESTROY,
                 0,
                 0 );

   }/* конец условия if(iOpenResult!=SP_NO_ERRORS) */
```

Разрешение использования мыши

При воспроизведении звукового файла может быть запрещено или разрешено использование мыши (для некоторых типов компьютеров качество воспроизведения звука при перемещении курсора мыши может несколько ухудшаться).

В функции WinMain() мы разрешили мышь с помощью функции sp_EnableMouseDuringPlay():

```
/*---------------------------------------
 Разрешаем работу мыши при воспроизведении.
 ---------------------------------------*/
sp_EnableMouseDuringPlay ();
```

Прототип функции sp_EnableMouseDuringPlay() объявлен в файле c:\spSDK\TegoWlib\sp4Win.h как

```
int sp_EnableMouseDuringPlay ( void )
```

Возвращаемое функцией sp_EnableMouseDuringPlay() значение не используется.

Примечание:

Функция sp_EnableMouseDuringPlay() разрешает использование мыши во время воспроизведения.

Параметры: Отсутствуют.

Возвращаемое значение: Целое число. Не используется.

Инициализация перед выполнением цикла обработки сообщений

В Generic2 мы объявили глобальную переменную giPlayEnable как

```
int giPlayEnable;
```

Перед выполнением цикла обработки сообщений в WinMain() устанавливаем переменную giPlayEnable в 1:

```
/*-----------------------------------------------
 Устанавливаем переменную giPlayEnable таким образом,
 чтобы разрешить воспроизведение.
 -----------------------------------------------*/
giPlayEnable = 1;
```

Эта переменная проверяется позднее внутри цикла обработки сообщений.

Подобным образом мы объявили глобальную переменную glCurrentBackGndPlay в Generic2.h как

```
int glCurrentBackGndPlay;
```

Эта переменная используется в цикле обработки сообщений как счетчик положения текущего байта фонового звукового файла. При инициализации этой переменной мы указали значение 0, потому что хотим начать воспроизведение звукового файла с начала:

```
/*-----------------------------------
 Устанавливаем счетчик сыгранного байта
 в начало звукового файла.
 -----------------------------------*/
glCurrentBackGndByte = 0L;
```

Воспроизведение звуковых файлов внутри цикла обработки сообщений

Ниже приведен цикл обработки сообщений Generic2:

```
/*-----------------------------------------
 Цикл обработки сообщений (типа Generic2).
 -----------------------------------------*/
while (TRUE)
      {
       if ( PeekMessage ( &msg, NULL, 0, 0, PM_REMOVE ) )
          {
          /*-----------------------------------------
          В очереди сообщений есть сообщение.
          Проверим, является ли это сообщение WM_QUIT.
          -----------------------------------------*/
      if ( msg.message == WM_QUIT )
          {
          /*-------------------------
          Принято сообщение WM_QUIT.
          -------------------------*/
```

```
            break; /* Завершаем приложение. */
            }
         /*-------------------------------
         Сообщение принято.
         Принятое сообщение — не WM_QUIT.
         -------------------------------*/
         TranslateMessage ( &msg );
         DispatchMessage  ( &msg );
         }
      else
         {
         /*--------------------------------
         В очереди сообщений сообщений нет.
         --------------------------------*/
         if ( giPlayEnable == 1 )
            {
            /*-------------------
            Можно воспроизводить.
            --------------------*/
            glCurrentBackGndByte =
            sp_PlayF ( glCurrentBackGndByte,
                       glCurrentBackGndByte + 2000L );
            }
         }
      }/* конец цикла while (TRUE) */
```

Фрагмент текста программы после ключевого слова else оператора if(PeekMessage()) служит для воспроизведения 2000 байт звукового файла. Этот фрагмент выполняется тогда, когда функция PeekMessage() обнаружит отсутствие сообщений для нашей программы. Функция sp_PlayF() выполняется, если значение переменной giPlayEnable равно 1. Эта переменная служит флагом, запрещающим или разрешающим воспроизведение.

Если переменная glCurrentBackGndByte изменяется таким образом, что функция sp_PlayF() выполняется опять, будут воспроизведены следующие 2000 байт.

Функция WndProc() программы Generic2

Функция WndProc() программы Generic2 не отличается от WndProc() программы Generic1. В варианте WM_CREATE мы получаем lpfn диалогового окна **О программе Generic2**, а в варианте WM_COMMAND — обрабатываем команду меню.

В вариантах IDM_ENABLE_PLAY (**Разрешить воспроизведение**) и IDM_DISABLE_PLAY (**Запретить воспроизведение**) значение переменной giPalyEnable также изменяется:

```
case IDM_ENABLE_PLAY:
     giPlayEnable = 1 ;
     return 0L;

case IDM_DISABLE_PLAY:
     giPlayEnable = 0 ;
     return 0L;
```

Обратите внимание на то, что переменная giPlayEnable используется в цикле обработки сообщений функции WinMain().

В вариантах IDM_ENABLE_MOUSE (**Разрешить мышь**) и IDM_DISABLE_MOUSE (**Запретить мышь**) выполняются функции sp_EnableMouseDuringPlay() и sp_DisableMouse-DuringPlay() соответственно:

```
case IDM_ENABLE_MOUSE:
     /*-------------------------------------
     Разрешим работу мыши при воспроизведении.
     -------------------------------------*/
```

```
        sp_EnableMouseDuringPlay ();
        return 0L;

    case IDM_DISABLE_MOUSE:
        /*---------------------------------------
        Запретим работу мыши при воспроизведении.
        ---------------------------------------*/
        sp_DisableMouseDuringPlay ();
        return 0L;
```

Запрет использования мыши

Можно запретить использовать мышь во время воспроизведения, если перед вызовом функции sp_PlayF() вызвать функцию sp_DisableMouseDuringPlay():

```
/*---------------------------------------
Запретим работу мыши при воспроизведении.
---------------------------------------*/
sp_DisableMouseDuringPlay ();
```

Прототип функции sp_DisableMouseDuringPlay() объявлен в файле c:\spSDK\TegoWlib\sp4Win.h как

```
int sp_DisableMouseDuringPlay ( void );
```

Возвращаемое sp_DisableMouseDuringPlay() значение не используется.

Примечание:

Функция sp_DisableMouseDuringPlay() устанавливает запрет использования мыши при воспроизведении.

Параметры: Отсутствуют.

Возвращаемое значение: Целое число. Не используется.

Выполнение нескольких экземпляров программы

Поскольку мы не ограничивали количество экземпляров программы Generic2, которые могут выполняться одновременно, то можно вызвать не один экземпляр этой программы. Первый раз запустите Generic2, разрешив использование мыши, а затем переключитесь в Диспетчер Программ Windows и вызовите программу Generic2 второй раз.

Вы услышите два фоновых звуковых файла, воспроизводимых одновременно. Однако поскольку фрагменты имеют объем 2000 байт, то звуковые файлы воспроизводятся несинхронно. Для того чтобы добиться максимальной синхронности, необходимо уменьшить размер фрагмента, воспроизводимого в цикле обработки сообщений.

Используя многозадачность и возможность одновременного выполнения нескольких экземпляров программы, можно создать громадное количество интереснейших программ. В частности, вы можете задать флаг для определения экземпляра программы и для разных экземпляров воспроизводить различные фрагменты звукового файла.

Программа SayName

Программа SayName — это программа типа Generic2, которая работает в фоновом режиме и осуществляет мониторинг (текущий контроль) сеанса Windows. Определенные события, возникающие во время сеанса Windows, SayName сопровождает комментариями. Так, если пользователь вызывает определенную программу Windows, программа SayName объявляет ее имя. При выходе из Windows программа SayName выдает пользователю звуковое сообщение: *Are you sure you want to quit Windows?* (Вы уверены в том, что хотите выйти из Windows?)

Компиляция и компоновка программы SayName компилятором фирмы Microsoft

Для компиляции и компоновки программы SayName компилятором фирмы Microsoft необходимо выполнить следующие действия:

- Убедиться в том, что ваш компьютер находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4Win.
- После приглашения DOS ввести:

```
NMAKE SayName.mak    [Enter]
```

Для компиляции и компоновки программы SayName компилятором фирмы Borland выполните такие действия:

- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
MAKE -f SayName.bmk    [Enter]
```

Выполнение программы SayName

Для выполнения программы SayName необходимо:

- Из меню **Файл** Диспетчера Программ Windows выбрать команду **Выполнить**.
- В диалоговом окне **Пролистать** выбрать файл

```
c:\spSDK\samp4Win\SayName.exe
```

Появляется главное окно, приведенное на рис. 9.4.

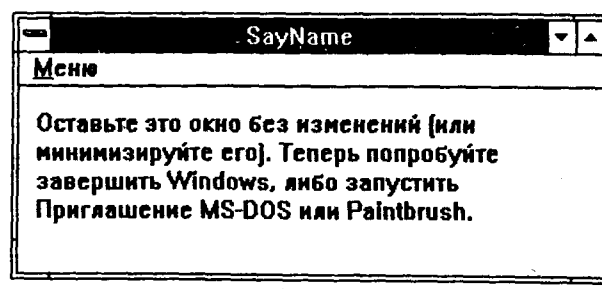


Рис. 9.4. Главное окно программы SayName

Меню программы SayName (см. рис. 9.5) содержит две команды — **Завершить** и **О Программе**. Диалоговое окно **О Программе SayName** изображено на рис. 9.6.

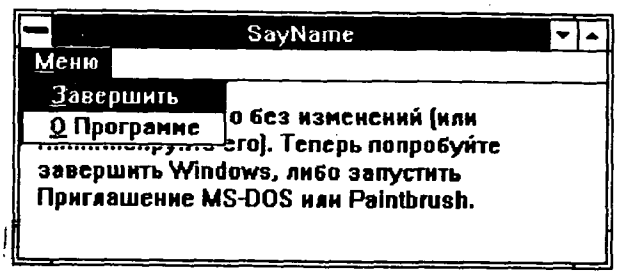


Рис. 9.5. Меню программы SayName

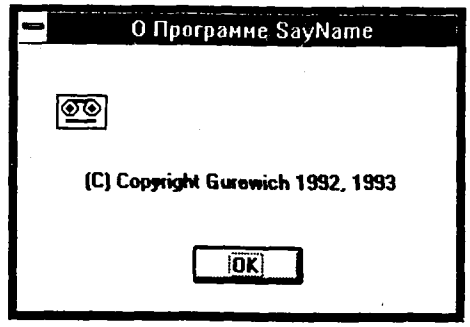


Рис. 9.6. Диалоговое окно О Программе SayName

После запуска программа SayName начинает контролировать состояние Windows, пытаясь обнаружить определенные события. События, на которые может реагировать программа SayName, и действия программы при их обнаружении приведены в следующей таблице:

Событие	*Действие, предпринимаемое SayName*
Пользователь запускает Paintbrush первый раз	SayName воспроизводит звуковое приглашение: Welcome to Paintbrush
Переход в режим MS-DOS	SayName воспроизводит звуковое приглашение: *Welcome to MS-DOS*
Пользователь требует завершения Windows	SayName воспроизводит звуковой запрос: *Are you sure you want to quit Windows?*

Файлы программы SayName

Программа SayName состоит из следующих файлов:

SayName.h	#include-файл программы
SayName.c	Текст программы на C
SayName.rc	Файл ресурсов программы
SayName.def	Файл определения модуля программы
SayName.mak	Make-файл программы
Tape.ico	Пиктограмма программы

Эти файлы расположены на вашем винчестере в директории c:\spSDK\Samp4Win\. Все файлы программы SayName приведены в листингах 9.6–9.10.

Листинг 9.6. SayName.h

```
/*=======================================
 ИМЯ ФАЙЛА: SayName.h

 (C) Copyright Gurewich 1992, 1993
 =======================================*/

/*--------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc          ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export AboutDlgProc     ( HWND, UINT, UINT, LONG ) ;

void  AnnounceTheApplication ( void );

/*------
 #define
 -------*/
#define IDM_QUIT      1  /* Команда меню  Завершить    */
#define IDM_ABOUT     2  /* Команда меню  О программе */

/*--------------------
 Глобальные переменные
 --------------------*/
char       gszAppName[] = "SayName" ;  /* Имя нашего приложения. */
HINSTANCE  ghInst;                     /* текущий экземпляр.     */
```

Листинг 9.7. SayName.c

```
/*============================================================
 ПРОГРАММА: SayName.c
 ----------
 (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

 НАЗНАЧЕНИЕ ПРОГРАММЫ:
 ---------------------
 Это программа типа Generic 2.                                \

 Эта программа воспроизводит приглашающее сообщение, когда
 пользователь запускает определенное приложение Windows, и
 воспроизводит предупреждающее сообщение, когда пользователь
 завершает работу с Windows.

 ============================================================*/
/*-------
 #include
 --------*/
/*---------------------------------------------
 windows.h требуется для всех приложений Windows.
 ---------------------------------------------*/
#include <windows.h>
```

```
/*-------------------------------------------------------
 sp4Win.h требуется для того, чтобы в этих приложениях
 могли быть использованы функции sp_, макросы SP_ и
 #define из библиотеки звуковой поддержки.
 -------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-------------------------------------------------------
 Определения и прототипы, характерные для этого приложения.
 -------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\SayName.h"
/*-------------------------------------------------------
 Для функций C, требующих стандартные #include-файлы C.
 -------------------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>

/*==================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*-----------------------------------
 Локальные и статические переменные.
 -----------------------------------*/
HWND      hWnd;  /* Идентификатор окна нашего приложения.    */
MSG        msg;  /* Сообщение, которое будет обрабатываться */
                 /*  нашим приложением.                      */
WNDCLASS    wc;  /* Класс окна нашего приложения.           */

int       iOpenResult; /* Результат открытия звукового сеанса. */

/*---------------------------------------
 Сделаем hInstance глобальной переменной.
 ---------------------------------------*/
ghInst = hInstance;

/*-----------------------------------------------
 Обновляем структуру класса окна и регистрируем
 класс окна.
 -----------------------------------------------*/
if ( !hPrevInstance )
   {
   /*---------------------------------------
   Условие if удовлетворено, это самый первый
   запуск данного приложения.
   ---------------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfTape" ) ;
```

```
    wc.hCursor       = LoadCursor ( NULL, IDC_ARROW ) ;
    wc.hbrBackground = GetStockObject ( WHITE_BRUSH ) ;
    wc.lpszMenuName  = gszAppName ;
    wc.lpszClassName = gszAppName ;

   /*----------------
   Регистрируем окно.
   ------------------*/
   RegisterClass ( &wc );

}/* конец условия if( !hPrevInstance ) */

/*-------------------------
 Открываем звуковой сеанс.
 -------------------------*/
iOpenResult =
sp_OpenSession ( "c:\\spSDK\\TSfiles\\SayName.ts",
                 SP_NON_STAND_ALONE,
                 0L,
                 SP_TS_TYPE );

/*-----------------------------------------------
 Завершаем работу, если невозможно открыть файл.
 -----------------------------------------------*/
if ( iOpenResult != SP_NO_ERRORS )
   {
   MessageBox ( NULL,
                "Невозможно открыть звуковой сеанс!",
                "Сообщение",
                MB_ICONINFORMATION );

   /*-------------------
   Завершаем приложение.
   ---------------------*/
   SendMessage ( hWnd,
                 WM_DESTROY,
                 0,
                 0 );

   }/* конец условия if(iOpenResult!=SP_NO_ERRORS) */

/*------------------------------
 Создаем окно нашего приложения.
 ------------------------------*/
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      350,
                      150,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );
```

```
/*-------------------------------------------
 Показываем и обновляем окно нашего приложения.
 -------------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );

/*---------------------------------------
 Цикл обработки сообщений (типа Generic2).
 ---------------------------------------*/
while (TRUE)
      {
      if ( PeekMessage ( &msg, NULL, 0, 0, PM_REMOVE ) )
         {
         /*-----------------------------------------
         В очереди сообщений существует сообщение.
         Проверим, является ли это сообщение WM_QUIT.
         -----------------------------------------*/
         if ( msg.message == WM_QUIT )
            {
            /*-------------------------
            Принято сообщение WM_QUIT.
            -------------------------*/
            break; /* Завершаем приложение. */
            }
         /*-------------------------------
         Сообщение принято.
         Принятое сообщение — не WM_QUIT.
         -------------------------------*/
         TranslateMessage ( &msg );
         DispatchMessage  ( &msg );
         }
      else
         {
         /*--------------------------------
         В очереди сообщений сообщений нет.
         --------------------------------*/
         AnnounceTheApplication();
         }
      }/* конец цикла while (TRUE) */

return msg.wParam ;

} /* Конец функции. */
/*========================= конец WinMain() ====================*/

/*==================
 ФУНКЦИЯ: WndProc()
 ==================*/
/*------------------------------
 НАЗНАЧЕНИЕ: Обработка сообщений.
 ------------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
```

```
{
/*-----------------------------------
 Локальные и статические переменные.
 -----------------------------------*/
HDC           hdc;      /* Необходима для отображения текста. */
PAINTSTRUCT   ps;       /* Необходима для отображения текста. */
RECT          rect;     /* Необходима для отображения текста. */

static FARPROC lpfnAboutDlgProc ; /* Для диалогового окна О Программе. */

switch ( message )
       {
       case WM_CREATE:
            /*-----------------------------------------------------
            Получаем lpfnAboutDlgProc диалогового окна О Программе.
            -----------------------------------------------------*/
            lpfnAboutDlgProc =
            MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);

            return 0;

      case WM_PAINT:
            hdc = BeginPaint ( hWnd, &ps );
            GetClientRect ( hWnd, &rect );
            SetRect ( &rect,
                      rect.left+10,
                      rect.top+10,
                      rect.right-10,
                      rect.bottom);
            DrawText ( hdc,
"Оставьте это окно без изменений (или \
минимизируйте его). Теперь попробуйте \
завершить Windows, либо запустить     \
Приглашение MS-DOS или Paintbrush.",
                      -1,
                      &rect,
                      DT_LEFT | DT_WORDBREAK  );
            EndPaint ( hWnd, &ps );
            return 0;
      case WM_COMMAND:
            SetFocus ( hWnd );
            /*-------------------------------------
            Пользователь выбрал команду из меню или
            нажал командную кнопку.
            -------------------------------------*/

            switch (wParam)
                   {
                   case IDM_QUIT:
                        /*-----------------------------------
                        Пользователь выбрал команду Завершить.
                        -----------------------------------*/
                        DestroyWindow (hWnd);
                        return 0L;
```

```
                    case IDM_ABOUT :
                         /*-------------------------------------
                         Пользователь выбрал команду О Программе.
                         -------------------------------------*/
                         DialogBox ( ghInst,
                                     "AboutBox",
                                     hWnd,
                                     lpfnAboutDlgProc );
                         return 0;

                    }/* конец оператора switch(wParam)
                        в варианте WM_COMMAND */
          case WM_DESTROY:
               PostQuitMessage (0);
               return 0;
          }/* конец оператора switch (message) */

/*-----------------------
 Сообщение не обработано.
 -----------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}
/*================== конец WndProc() ====================*/


/*=======================
 ФУНКЦИЯ: AboutDlgProc()
 =======================*/
/*-----------------------------------------
 НАЗНАЧЕНИЕ:
 Это процедура диалогового окна О Программе.
 -----------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;

       case WM_COMMAND :
            switch ( wParam )
                   {
                   case IDOK :
                   case IDCANCEL :
                        EndDialog ( hDlg, 0 );
                        return TRUE;
                   }
       }
return FALSE ;

}/* Конец функции. */
/*================ Конец AboutDlgProc() ==============*/
```

```
/*================================
 ФУНКЦИЯ: AnnounceTheApplication()
 ================================*/
void AnnounceTheApplication ( void )
{
HWND hWindow;

static iPaintbrush   = 0;
static iExitWindows  = 0;
static iDOSshell     = 0;

if ( iPaintbrush == 0 )
   {
   /*---------------------------
   Paintbrush только что запущен?
   ---------------------------*/

   hWindow = FindWindow ( NULL, "Paintbrush — (Untitled)" );
   if ( hWindow )
      {
      iPaintbrush = 1;
      sp_PlayF ( 0L, 18153L );
      }
   }
else
   {
   hWindow = FindWindow ( NULL, "Paintbrush — (Untitled)" );
   if ( !hWindow )
        iPaintbrush = 0;
   }

if ( iExitWindows == 0 )
   {
   /*-----------------------
   Запрос завершить Windows?
   -----------------------*/
   hWindow = FindWindow ( NULL, "Exit Windows") ;
   if ( hWindow )
      {
      iExitWindows = 1;
      sp_PlayF ( 18153L, 42672L );
      }
   }
else
   {
   hWindow = FindWindow ( NULL, "Exit Windows" );

   if ( !hWindow )
        iExitWindows = 0;
   }

if ( iDOSshell == 0 )
   {
   /*-------------------------------------
   Приглашение MS-DOS только что запущено?
   -------------------------------------*/
   hWindow = FindWindow ( NULL, "MS-DOS Prompt" );
```

```
    if ( hWindow )
        {
        iDOSshell = 1;
        sp_PlayF ( 42672L, SP_END_OF_FILE );
        }
    }
else
    {
    hWindow = FindWindow ( NULL, "MS-DOS Prompt" );
    if ( !hWindow )
         iDOSshell = 0;
    }

}/* Конец функции. */
/*============ Конец функции =============*/
```

Листинг 9.8. SayName.rc.

```
/*================================================
 ИМЯ ФАЙЛА: SayName.rc
 ----------

 НАЗНАЧЕНИЕ ПРОГРАММЫ:
 ---------------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 ================================================*/
/*-------
 #include
 --------*/
#include <windows.h>
#include "SayName.h"

/*---
 Меню
 ----*/
SayName  MENU
BEGIN
  POPUP  "&Меню"
    BEGIN
     MENUITEM "&Завершить",  IDM_QUIT
     MENUITEM "&О Программе", IDM_ABOUT
    END
END

/*--------------------------------
 Определение пиктограммы кассеты.
 Имя файла: Tape.ico
 Имя пиктограммы: IconOfTape
 --------------------------------*/
IconOfTape ICON Tape.ico
```

```
/*-------------------------
 Диалоговое окно О Программе.
 -------------------------*/

AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "О Программе SayName"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON    "OK", IDOK, 64, 75, 40, 14
    CTEXT         "(C) Copyright Gurewich 1992, 1993", -1,
                  13, 47, 137, 18
    ICON          "IconOfTape", -1, 14, 12, 18, 20
END
```

Листинг 9.9. SayName.def.

```
;=======================================
; Файл определения модуля для SayName.c
;=======================================

NAME        SayName

DESCRIPTION  'Программа SayName. (C) Copyright Gurewich 1992, 1993'

EXETYPE     WINDOWS

STUB        'WINSTUB.EXE'

CODE  PRELOAD MOVEABLE DISCARDABLE

DATA  PRELOAD MOVEABLE MULTIPLE

HEAPSIZE    1024
STACKSIZE   8192
```

Листинг 9.10. SayName.mak.

```
#=============
# SayName.mak
#=============

SayName.exe : SayName.obj SayName.h SayName.def SayName.res
    link /nod SayName.obj, SayName.exe, NUL, \
       slibcew.lib oldnames.lib libw.lib commdlg \
       c:\spSDK\TegoWlib\TegoWin.lib, \
       SayName.def
    rc -t SayName.res

SayName.obj : SayName.c SayName.h
   cl -c -G2sw -Ow -W3 -Zp SayName.c

SayName.res : SayName.rc SayName.h Tape.ico
   rc -r SayName.rc
```

Функция WinMain() программы SayName

В функции WinMain() мы открываем звуковой сеанс со звуковым файлом c:\spSDK\TSfiles\SayName.ts:

```
/*------------------------
 Открываем звуковой сеанс.
 ------------------------*/
iOpenResult =
sp_OpenSession ( "c:\\spSDK\\TSfiles\\SayName.ts",
                 SP_NON_STAND_ALONE,
                 0L,
                 SP_TS_TYPE );
/*-------------------------------------------
 Завершаем работу, если невозможно открыть файл.
 -------------------------------------------*/
if ( iOpenResult != SP_NO_ERRORS )
   {
   MessageBox ( NULL,
                "Невозможно открыть звуковой сеанс!",
                "Сообщение",
                MB_ICONINFORMATION );

   /*-------------------
   Завершаем приложение.
   --------------------*/
   SendMessage ( hWnd,
                 WM_DESTROY,
                 0,
                 0 );

   }/* конец условия if(iOpenResult!=SP_NO_ERRORS) */
```

Звуковой файл SayName.ts состоит из трех фрагментов. Поскольку предполагается, что у вас нет редактора Sound Editor фирмы TS, то ниже приведена структура SayName.t.

От байта	*До байта*	*Фраза*
0	18153	Welcome to Paintbrush
18153	42672	Are you sure you want to quit Windows?
42672L	SP_END_OF_FILE	Welcome to MS-DOS

Цикл обработки сообщений программы SayName

Так как SayName является программой типа Generic 2, цикл обработки сообщений у нее такой же, как и у Generic2:

```
/*---------------------------------------
 Цикл обработки сообщений (типа Generic2).
 ---------------------------------------*/
while (TRUE)
      {
      if ( PeekMessage ( &msg, NULL, 0, 0, PM_REMOVE ) )
         {
         /*-----------------------------------
         В очереди сообщений есть сообщение.
         Проверим, не является ли оно WM_QUIT.
         -----------------------------------*/
```

```
            if ( msg.message == WM_QUIT )
                {
                /*-------------------------
                Принято сообщение WM_QUIT.
                -------------------------*/
                break; /* Завершаем приложение. */
                }
            /*-------------------------------
            Сообщение принято.
            Принятое сообщение — не WM_QUIT.
            -------------------------------*/
        TranslateMessage ( &msg );
        DispatchMessage  ( &msg );
        }
    else
        {
        /*-------------------------------
        В очереди сообщений нет сообщений.
        -------------------------------*/
         AnnounceTheApplication();
        }
    }/* конец цикла while (TRUE) */
```

Всякий раз, когда сообщения для программы SayName отсутствуют, выполняется функция AnnounceTheApplication().

Контроль сеанса Windows

Контроль сеанса Windows осуществляется функцией AnnounceTheApplication(). Так как программа SayName — это программа типа Generic2, и поскольку мы вызываем функцию AnnounceTheApplication() из цикла обработки сообщений функции WinMain(), то функция AnnounceTheApplication() выполняется постоянно, даже если пользователь работает с другими программами.

В функции AnnounceTheApplication() определены три статические переменные:

```
static iPaintbrush    = 0;
static iExitWindows   = 0;
static iDOSshell      = 0;
```

Эти переменные используются как флаги для указания того, запущена конкретная программа или нет. Например, если запущена программа Paintbrush, то флаг iPaintbrush должен быть установлен в 1.

Объявляя данные статические переменные, мы установили их в 0 (и таким образом предположили, что ни одна из этих программ еще не запущена).

Назначение функции AnnounceTheApplication() заключается в том, что она сообщает о наступлении некоторого события, например, о запуске программы. Так, если только что запущена программа Paintbrush, то функция AnnounceTheApplication() должна воспроизводить звуковое приглашение *Welcome to Paintbrush.*

Тот факт, что данная программа только что запущена, определяется выполнением двух условий:

1. Флаг программы равен 0.
2. Функция FindWindow() указывает, что программа запущена.

Например, для проверки того, запущена ли программа Paintbrush, мы анализируем два условия, используя следующие вложенные операторы if:

```
if ( iPaintbrush == 0 )
    {
    /*----------------------------
    Paintbrush только что запущен?
    ----------------------------*/
    hWindow = FindWindow ( NULL, "Paintbrush — (Untitled)" );
    if ( hWindow )
        {
        iPaintbrush = 1;
        sp_PlayF ( 0L, 18153L );
        }
    }
else
    {
    hWindow = FindWindow ( NULL, "Paintbrush — (Untitled)" );
    if ( !hWindow )
    iPaintbrush = 0;
    }
```

После обнаружения того факта, что программа Paintbrush только что запущена пользователем, мы устанавливаем флаг iPaintbrush в 1 и воспроизводим звуковое приглашение *Welcome to Paintbrush.*

Если условие else оператора if(iPaintbrush==0) выполнено, то программа Paintbrush была уже запущена и пользователь только что завершил ее. В этом случае мы устанавливаем флаг iPaintbrush в 0.

Подобным образом используем операторы if-else для обнаружения того, что произошел переход в режим MS-DOS или что пользователь пытается завершить сеанс работы с Windows.

Программа Organ

Программа Organ также является программой типа Generic2. Как видно из ее названия, эта программа имитирует игру на органе.

Компиляция, компоновка и выполнение программы Organ

Для компиляции и компоновки программы Organ компилятором фирмы Microsoft выполните следующие действия:

- Удостоверьтесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
NMAKE Organ.mak    [Enter]
```

Для компиляции и компоновки программы Organ компилятором фирмы Borland необходимо:

- Войти в директорию c:\spSDK\Samp4Win.
- После приглашения DOS ввести:

```
MAKE -f Organ.bmk    [Enter]
```

Для выполнения программы Organ:

- Выберите из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- В диалоговом окне **Пролистать** выберите файл

```
c:\spSDK\samp4Win\Organ.exe
```

Появляется главное окно, приведенное на рис. 9.7.

Меню программы Organ (см. рис. 9.8) содержит три команды — **Завершить**, **О Программе** и **Инструкции**. Диалоговое окно **О Программе Орган** показано на рис. 9.9, а диалоговое окно **Инструкции** — на рис. 9.10. Это окно содержит командную кнопку **Звуковые Инструкции**, после нажатия на которую пользователь получает соответствующие звуковые инструкции.

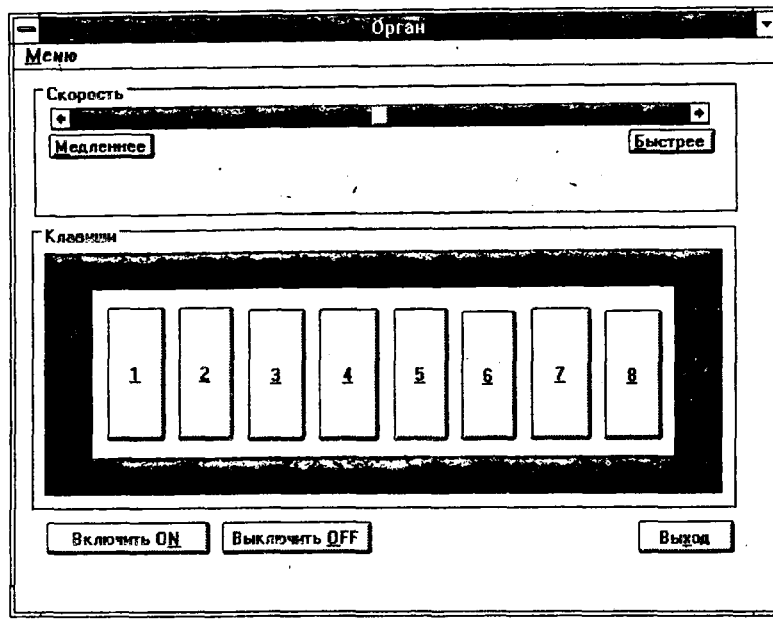


Рис. 9.7. Главное окно программы Organ

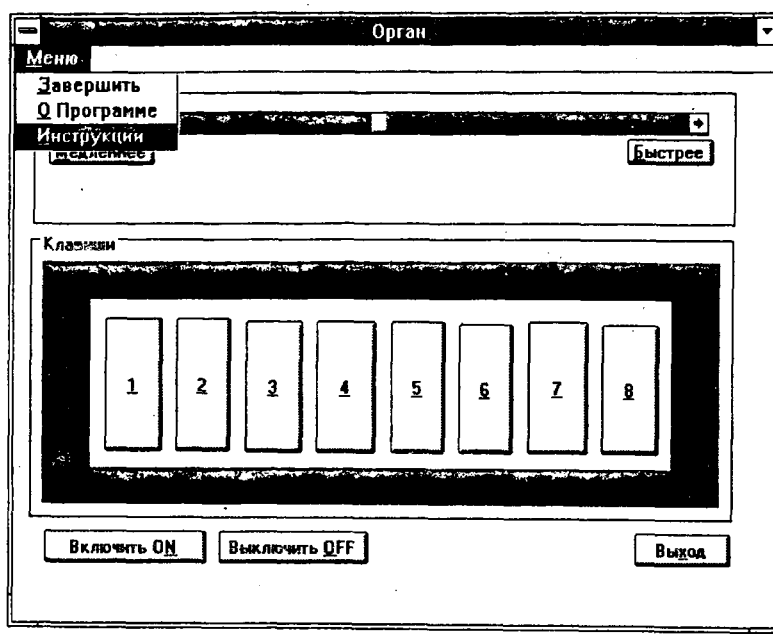


Рис. 9.8. Меню программы Organ

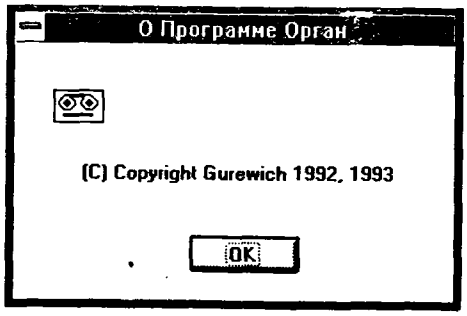


Рис. 9.9. Диалоговое окно программы Organ

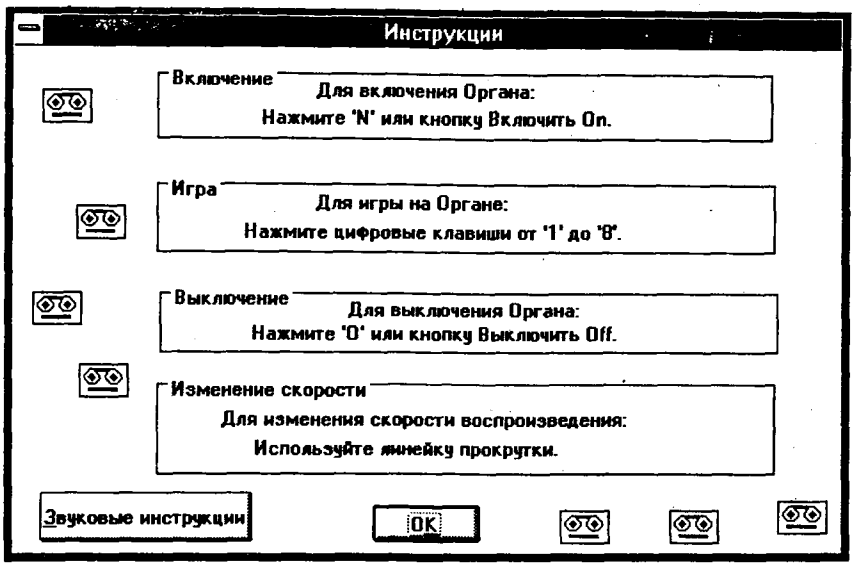


Рис.9.10. Диалоговое окно Инструкции программы Organ

В главном окне программы отображается клавиатура органа (см. рис. 9.7). Для включения органа нажмите клавишу [N] или командную кнопку **Включить ON**, для выключения — клавишу [O] или командную кнопку **Выключить OFF**.

Чтобы воспроизвести аккорды органа, следует нажимать цифровые клавиши от 1 до 8.

Пока орган включен, вы можете изменять скорость воспроизведения, нажимая клавишу [Б] для увеличения скорости или клавишу [М] — для ее уменьшения.

Файлы программы Organ

Программа Organ состоит из следующих файлов:

Organ.h	#include-файл программы
Organ.c	Текст программы на С
Organ.rc	Файл ресурсов программы
Organ.def	Файл определения модуля программы
Organ.mak	Make-файл программы
Tape.ico	Пиктограмма программы

Эти файлы записаны на вашем винчестере в директории c:\spSDK\Samp4Win\.

Функция WinMain() программы Organ

Функция WinMain() открывает звуковой сеанс со звуковым файлом c:\spSDK\TSfiles\Notes.ts:

```
iOpenOrganResult =
sp_OpenSession ( "c:\\spSDK\\TSfiles\\Notes.ts",
                 SP_NON_STAND_ALONE,
                 0L,
                 SP_TS_TYPE );
/*-------------------------------------------
 Завершаем работу, если невозможно открыть файл.
 -------------------------------------------*/
if ( iOpenOrganResult != SP_NO_ERRORS )
   {
   MessageBox ( NULL,
                "Невозможно открыть звуковой сеанс!",
                "Сообщение от WinMain()",
                MB_ICONINFORMATION );

   /*--------------------
   Завершаем приложение.
   --------------------*/
   SendMessage ( hWnd,
                 WM_DESTROY,
                 0,
                 0 );

   }/* конец условия if() */
```

Звуковой файл Notes.ts включает в себя все звуковые фрагменты, которые необходимы для программы Organ. Поскольку предполагается, что у вас нет редактора Sound Editor фирмы TS, ниже приводится разбивка данного файла на звуковые фрагменты.

От байта	*До байта*	*Звуковой фрагмент*
0	19049	Фоновая мелодия
19049	38606	Нота До органа
38606	57603	Нота Ре органа
57603	76560	Нота Ми органа
76560	95435	Нота Фа органа
95435	114074	Нота Соль органа
114074	130791	Нота Ля органа
130791	147624	Нота Си органа
147624	167681	Нота До второй октавы органа
167681	(конец файла)	Звуковые инструкции

Цикл обработки сообщений программы Organ

Цикл обработки сообщений программы Organ — это цикл типа Generic2.

```
while (TRUE)
   {
   if ( PeekMessage ( &msg, NULL, 0, 0, PM_REMOVE ) )
```

```
    {
    /*--------------------------------
    В очереди сообщений есть сообщение.
    Проверим, является ли оно WM_QUIT.
    --------------------------------*/
    if ( msg.message == WM_QUIT )
       {
       /*-----------------------
       Принято сообщение WM_QUIT.
       -----------------------*/
       break; /* Завершаем приложение. */
       }
    /*-----------------------------
    Сообщение принято.
    Принятое сообщение — не WM_QUIT.
    -----------------------------*/
    TranslateMessage ( &msg );
    DispatchMessage  ( &msg );
    }
 else
    PlayBackground ();
 }/* конец цикла while (TRUE) */
```

Когда сообщений для программы нет, выполняется функция PlayBackground().

Функция PlayBackground()

Функция PlayBackground() воспроизводит фоновую органную мелодию фрагментами по 2000 байт. После каждого вызова PlayBackground() воспроизводятся следующие 2000 байт. Мы выполняем воспроизведение только тогда, когда флаг giOrganIsOn равен 1:

```
if ( giOrganIsOn == 1)
   {
   /*-------------------------------------------
   Так как орган включен, играем фоновую музыку.
   -------------------------------------------*/
   glCurrentBackGndByte =
   sp_PlayF ( glCurrentBackGndByte,
              glCurrentBackGndByte + 2000L );

   if ( glCurrentBackGndByte >= 18000L )
      {
      sp_PlayF ( glCurrentBackGndByte, 19049L );
      glCurrentBackGndByte = 0L;
      }
   }
```

Фоновая мелодия записана в диапазоне от 0-го до 19049-го байта включительно. В функции WinMain() мы установили glCurrentBackGndByte в 0, так что первый раз воспроизведение начинается с байта 0.

Каждый раз, когда вызывается функция PlayBackground(), воспроизводится новый фрагмент в 2000 байт. Благодаря тому, что переменная glCurrentBackGndByte объявлена как глобальная (общедоступная), ее значение сохраняется в течение всего "периода жизни" программы.

Поскольку воспроизведение осуществляется фрагментами по 2000 байт, то при помощи оператора if мы определяем, когда переменная glCurrentBackGndByte достигнет значения 18000, и воспроизводим оставшийся звуковой фрагмент (от байта с номером 18000 до байта с номером 19049).

Инициализация линейки прокрутки

Так как программа Organ содержит линейку прокрутки, то необходимо задать диапазон этой линейки и установить ее бегунок в начальное положение. Это производится в варианте WM_PAINT вызовом функции SetNoteScrollBar():

```
case WM_PAINT:
     BeginPaint ( hWnd, &ps );
     SetNotesScrollBar ( hWnd );
     EndPaint ( hWnd, &ps );
     return 0;
```

Реакция на изменения положения бегунка линейки прокрутки

В варианте WM_HSCROLL обращаем внимание на изменения линейки прокрутки. Если пользователь нажал на кнопки со стрелками на концах линейки прокрутки или переместил бегунок, то мы соответствующим образом изменяем значение переменной giNoteSpeedScroll и выполняем функцию ChangeNotesSpeedRequest().

Обработка команд, вызванных с клавиш клавиатуры

В варианте WM_CHAR обрабатываем сообщения, которые были сгенерированы в ответ на нажатие клавиш на клавиатуре. Функция toupper() преобразует принятый wParam символ в верхний регистр, а затем производится анализ этого символа.

Завершение программы при нажатии клавиши [X]

Если пользователь нажал клавишу [X], то выполняется код для варианта 'x', вызывающий завершение программы:

```
switch ( wParam )
       {
       case 213: /* Код русской буквы X */
            /*--------------------------------
            Пользователь нажал [X] для выхода.
            --------------------------------*/
            DestroyWindow (hWnd);
            return 0L;
```

Включение и выключение органа

Если пользователь нажимает клавиши [N] или [O], то изменяется флаг giOrganIsOn:

```
case 'N':
     /*-----------------------------------------
     Пользователь нажал [N] для включения Органа.
     -----------------------------------------*/
     giOrganIsOn = 1;
     return 0L;

case 'O':
     /*-----------------------------------------
     Пользователь нажал [O] для выключения Органа.
     -----------------------------------------*/
     giOrganIsOn = 0;
     return 0L;
```

Ускорение и замедление воспроизведения

Пока орган играет (т.е. он включен), пользователь может увеличивать скорость воспроизведения, нажимая клавишу [Б]:

```
case 193: /* Код русской буквы Б */
     /*-----------------------------------
     Пользователь нажал [Б] для ускорения
     воспроизведения (Быстрее).
     -----------------------------------*/
     giNotesSpeedScroll++;
     ChangeNotesSpeedWasRequested ( hWnd );
     return 0L;
```

Замедлять воспроизведение он может, нажимая клавишу [М]:

```
case 204: /* Код русской буквы М */
     /*-----------------------------------
     Пользователь нажал [М] для замедления
     воспроизведения (Медленнее).
     -----------------------------------*/
     giNotesSpeedScroll--;
     ChangeNotesSpeedWasRequested ( hWnd );
     return 0L;
```

Игра на органе

Для игры на органе нажимают клавиши [1]–[8], которые соответствуют нотам До, Ре, Ми, Фа, Соль, Ля, Си и До второй октавы.

Для воспроизведения ноты выполняется соответствующая функция. Например, для воспроизведения ноты До выполняется функция PlayDo():

```
case '1':
     /*-----------------------
     Пользователь нажимает [1]
     -----------------------*/
     if ( giOrganIsOn == 1 )
          PlayDO();
     return 0L;
```

Другие функции воспроизведения нот — это PlayRE(), PlayME(), PlayFA(), PlaySO(), PlayLA(), PlayTI() и PlayDO2nd().

В каждой из этих функций используется sp_PlayF() для воспроизведения соответствующего звукового фрагмента.

Обработка команд, вызванных при помощи командных кнопок

Командные кнопки обрабатываются в варианте WM_COMMAND функции WndProc().

В варианте FAST_NOTES_PB мы обрабатываем командную кнопку **Быстрее**. Нажатие этой кнопки приводит к увеличению скорости воспроизведения:

```
case FAST_NOTES_PB:
     /*-----------------------------------------------
     Пользователь нажал кнопку Быстрее для увеличения
     скорости воспроизведения.
     -----------------------------------------------*/
     giNotesSpeedScroll++;
     ChangeNotesSpeedWasRequested ( hWnd );
     return 0L;
```

Аналогичным образом обрабатываем сообщения SLOW_NOTES_PB, TURN_ON_PB и TURN_OFF_PB.

Диалоговое окно Инструкции

Диалоговое окно **Инструкции** содержит командную кнопку **Звуковые Инструкции**, при нажатии на которую пользователь может получить соответствующие звуковые инструкции.

Поскольку перед открытием этого окна пользователь может изменить скорость воспроизведения, то мы должны установить нормальную скорость — с тем чтобы звуковые инструкции воспроизводились в виде естественной речи. В варианте AUDIO_PB устанавливаем скорость воспроизведения в нормальное значение, воспроизводим фрагмент инструкций, а затем восстанавливаем предыдущее значение скорости:

```
case AUDIO_PB:
     /*-----------------------------------------
     Пользователь нажал кнопку Звуковые Инструкции
     для получения звуковых инструкций.
     -----------------------------------------*/
     sp_SetNewSpeed ( MAX_NOTES_SPEED_SCROLL,
                      NAT_NOTES_SPEED_SCROLL,
                      NAT_NOTES_SPEED_SCROLL);

     sp_PlayF ( 167681L, SP_END_OF_FILE);

     sp_SetNewSpeed ( MAX_NOTES_SPEED_SCROLL,
                      NAT_NOTES_SPEED_SCROLL,
                      giNotesSpeedScroll);
     return TRUE;
```

Расширение программы Organ

При помощи программы Organ можно имитировать игру на простом органе. Конечно, можно улучшить эту программу, добавив в нее дополнительные возможности. Например, сейчас каждая нота воспроизводится "от начала до конца", т.е. пользователь не может изменить ее длительность. Чтобы предоставить ему такую возможность, воспроизводите ноты маленькими фрагментами, выполняя в интервалах между фрагментами функцию PeekMessage(), чтобы увидеть, отпустил ли пользователь данную клавишу (т.е. проверяется сообщение WM_KEYUP).

Программа Control2

В предыдущей главе мы рассматривали программу Controls, созданную на базе Generic1. Сейчас мы преобразуем Controls в программу на базе Generic2. Эта новая программа называется Control2.

Компиляция, компоновка и выполнение программы Control2

Для компиляции и компоновки программы Control2 компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
NMAKE Control2.mak    [Enter]
```

Для компиляции и компоновки программы Control2 компилятором фирмы Borland выполните следующие действия:

- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
MAKE -f Control2.bmk    [Enter]
```

Для выполнения программы Control2 необходимо:

- Выбрать из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- В диалоговом окне **Пролистать** выбрать файл

```
c:\spSDK\samp4Win\Control2.exe
```

Появляется главное окно, приведенное на рис. 9.11.

Меню программы Control2 (см. рис. 9.12) содержит четыре команды — **Завершить**, **О Программе**, **Разрешить мышь** и **Запретить мышь**. Диалоговое окно **О Программе** изображено на рис. 9.13.

Нажатие командной кнопки **Играть** приводит к воспроизведению звукового файла. Пока этот файл воспроизводится, на линейке прокрутки **Перемотка/Вперед** изменяется положение бегунка; изменяется и значение в поле **Позиция в секундах**.

Программа Control2 разрешает приостановить воспроизведение нажатием командной кнопки **Пауза** или клавиши [a].

Пользователь может воспроизводить звуковой файл в обратном направлении, нажав командную кнопку **Обратное воспроизведение** или клавишу [O].

Можно изменять скорость воспроизведения, используя линейку скорости или командные кнопки **Медленнее**, **Нормально** и **Быстрее**.

Пользователь может запускать программу с помощью мыши или без нее, нажимая в процессе воспроизведения клавиши [Alt+M], а затем выбирая соответствующую команду меню.

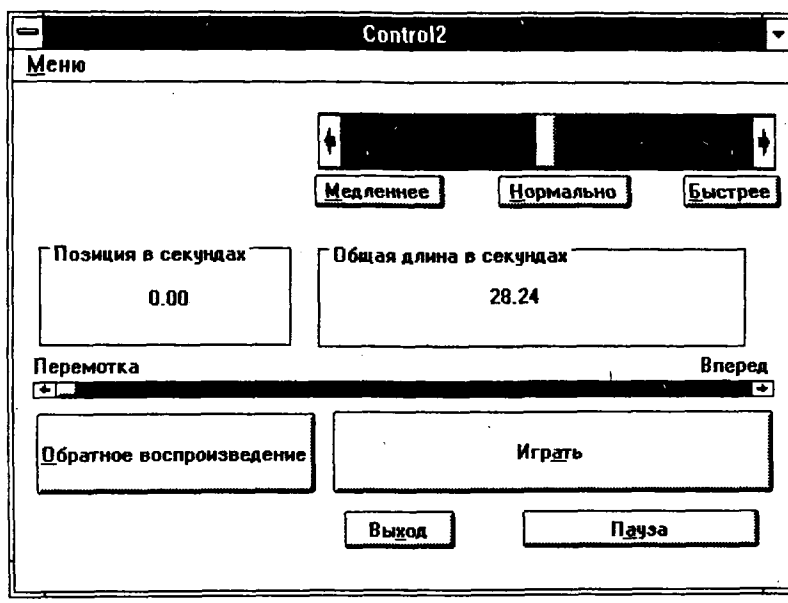


Рис.9.11. Главное окно программы Control2

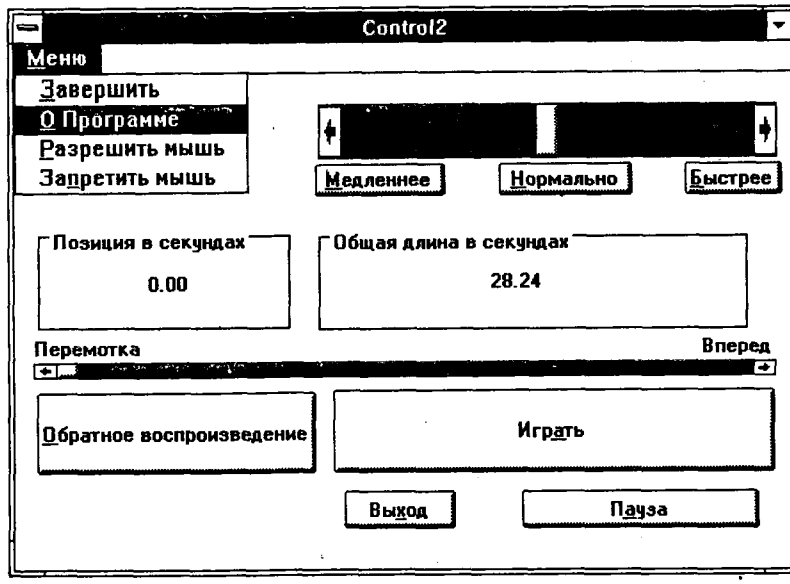


Рис. 9.12. Меню программы Control2

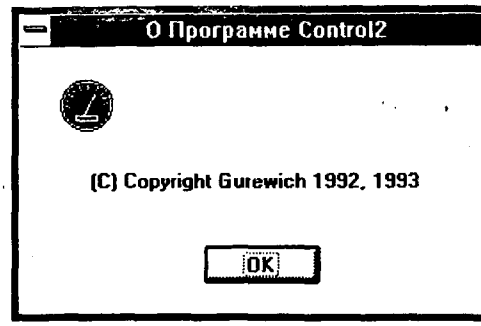


Рис. 9.13. Диалоговое окно О Программе Control2

Файлы программы Control2.

Программа Control2 состоит из следующих файлов:

Control2.h	#include-файл программы
Control2.c	Текст программы на C
Control2.rc	Файл ресурсов программы
Control2.def	Файл определения модуля программы
Control2.mak	Make-файл программы
Control2.ico	Пиктограмма программы

Эти файлы записаны на вашем винчестере в директории c:\spSDK\Samp4Win\.

Применение диалогового окна в качестве главного окна приложения

Программа Control2 использует диалоговое окно в качестве главного окна. Для этого необходимо установить значение элемента cbWndExtra структуры wc равным DLGWINDOWEXTRA:

```
wc.cbWndExtra = DLGWINDOWEXTRA ;
```

и созданием диалогового окна функцией CreateDialog(). Второй параметр в вызове CreateDialog() — Control2Box:

```
hWnd = CreateDialog ( ghInst, "Control2Box", 0, NULL);
```

Control2Box определен в файле Control2.rc как имя диалогового окна.

Открытие звукового сеанса

Звуковой сеанс открывается (как для неавтономной программы) со звуковым файлом типа S c:\spSDK\Sfiles\Day.s:

```
iOpenResult =
sp_OpenSession ( "c:\\spSDK\\Sfiles\\Day.s",
                 SP_NON_STAND_ALONE,
                 0L,
                 SP_S_TYPE );
/*-----------------------------------------------
 Завершаем работу, если невозможно открыть файл.
 -----------------------------------------------*/
if ( iOpenResult != SP_NO_ERRORS )
   {
   MessageBox ( NULL,
              "Невозможно открыть звуковой сеанс!",
              "Сообщение от Control2.exe",
               MB_ICONINFORMATION );

   /*-------------------
   Завершаем приложение.
   --------------------*/
   SendMessage ( hWnd,
                 WM_DESTROY,
                 0,
                 0 );

   }/* конец условия if(iOpenResult!=SP_NO_ERRORS) */
```

Чтобы при воспроизведении можно было использовать мышь, выполняем функцию sp_EnableMouseDuringPlay():

```
sp_EnableMouseDuringPlay()
```

Цикл обработки сообщений Control2

Цикл обработки сообщений Control2 — это цикл обработки сообщений типа Generic2. Функция ThePlay() выполняется каждый раз, когда сообщения для программы отсутствуют:

```
while (TRUE)
      {
      if ( PeekMessage ( &msg, NULL, 0, 0, PM_REMOVE ) )
```

```
            {
            /*---------------------------------
            В очереди сообщений есть сообщение.
            Проверим, является ли оно WM_QUIT.
            ---------------------------------*/

            if ( msg.message == WM_QUIT )
               {
               /*------------------------
               Принято сообщение WM_QUIT.
               ------------------------*/
               break; /* Завершаем приложение. */
               }
            /*--------------------------------
            Сообщение принято.
            Принятое сообщение — не WM_QUIT.
            --------------------------------*/
            TranslateMessage ( &msg );
            DispatchMessage  ( &msg );
            }
         else
            {
            /*---------------------------------
            В очереди сообщений нет сообщений.
            ---------------------------------*/
            ThePlay(hWnd);
            }
      }/* конец цикла while (TRUE) */
```

Функция ThePlay()

Функция ThePlay() воспроизводит звуковой файл фрагментами по 2000 байт. Мы пользуемся переменной giPlayEnable как флагом, показывающим, как должен воспроизводиться звуковой файл: в прямом направлении, в обратном направлении или не воспроизводиться вообще:

```
if ( giPlayEnable == 1 )
   {
   /*-------------------
   Можно воспроизводить.
   ---------------------*/
   glCurrentByte =
   sp_PlayF ( glCurrentByte,
              glCurrentByte + 2000L );
   DisplayLocation ( hWnd, glCurrentByte );
   }

if ( giPlayEnable == -1 )
   {
   /*-----------------------------------------
   Можно воспроизводить в обратном направлении.
   -----------------------------------------*/
   glCurrentByte =
   sp_PlayB ( glCurrentByte,
              glCurrentByte - 2000L );
   DisplayLocation ( hWnd, glCurrentByte );

   if ( glCurrentByte == 0 )
      {
      /*-----------
      Начало файла.
      -------------*/
```

```
        SendMessage ( hWnd,
                      WM_COMMAND,
                      PAUSE_PB,
                      0);
    }
```

После воспроизведения каждого фрагмента выполняется функция DisplayLocation(), которая изменяет положение бегунка линейки прокрутки **Перемотка/Вперед** и значение в текстовом окне **Позиция в секундах**.

Во время воспроизведения в прямом направлении звуковой файл воспроизводится в бесконечном цикле. Обратите внимание: когда достигнут конец файла, функция sp_PlayF() возвращает 0. Следовательно, после того как завершится последний фрагмент в 2000 байт, sp_PlayF() вернет 0, и glCurrentByte установится в 0, так что воспроизведение следующего фрагмента начнется от начала звукового файла.

Во время воспроизведения файла в обратном направлении при помощи оператора if(glCurrentByte==0) мы устанавливаем, когда достигнуто начало файла. В этом случае мы останавливаем воспроизведение, выполняя функцию SendMessage(), и посылаем сообщение WM_COMMAND с параметром wParam, равным PAUSE_PB, что равнозначно нажатию командной кнопки **Пауза**.

Вариант WM_CREATE программы Control2

В варианте WM_CREATE получаем lpfn диалогового окна **О Программе**, извлекаем размер звукового файла и инициализируем переменные, в которых хранится текущее положение бегунков линеек прокрутки **Скорость** и **Перемотка/Вперед**:

```
case WM_CREATE:
     /*-------------------------------------------------------
     Получаем lpfnAboutDlgProc диалогового окна О Программе.
     -------------------------------------------------------*/
     lpfnAboutDlgProc =
     MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);

     /*---------------------------------------
     Находим размер (в байтах) звукового файла.
     ---------------------------------------*/
     glFileSizeInBytes = sp_GetFileSizeInBytes ();

     /*---------------------------------------
     Переменная giPositionScroll изменилась.
     Эта переменная отражает положение
     бегунка линейки прокрутки Перемотка/Вперед.
     ---------------------------------------*/
     giPositionScroll = MIN_SPEED_SCROLL ;

     /*---------------------------------
     Переменная giSpeedScroll изменилась.
     Эта переменная отражает положение
     бегунка линейки прокрутки скорости.
     ---------------------------------*/
     giSpeedScroll = NAT_SPEED_SCROLL ;

     return 0;
```

Вариант WM_PAINT программы Control2

В варианте WM_PAINT мы отображаем в диалоговом окне длительность звучания файла в секундах, используя функцию DisplayFileSize(), затем, если нет воспроизведения, запрещаем использование командной кнопки **Пауза**, после чего инициализируем линейки прокрутки, используя функцию InitScrollBar():

```
case WM_PAINT:
     BeginPaint ( hWnd, &ps );
     /*-----------------------------------
     Определяем и отображаем длительность
     звучания файла (в секундах).
     -----------------------------------*/
     DisplayFileSize ( hWnd );

     if ( giPlayEnable == 0 )
        {
        /*---------------------
        Запрещаем кнопку Пауза.
        ---------------------*/
        hWndCtrl = GetDlgItem ( hWnd, PAUSE_PB );
        EnableWindow ( hWndCtrl, 0 );
        }

     /*-------------------------------
     Инициализируем линейки прокрутки.
     -------------------------------*/
     InitScrollBars ( hWnd );

     EndPaint ( hWnd, &ps );
     return 0;
```

Вариант WM_CHAR программы Control2

В варианте WM_CHAR обрабатываются все сообщения, полученные в результате нажатия пользователем клавиш клавиатуры. Поскольку каждая горячая клавиша закреплена за соответствующей командной кнопкой, то программа реагирует на нажатие этих клавиш, используя функцию SendMessage(), т.е. мы посылаем сообщения от командных кнопок, которые соответствуют нажатой клавише. Например, когда мы обнаруживаем, что пользователь нажал клавишу [O], то посылаем сообщение WM_COMMAND с параметром wParam, равным PLAY_BACK_PB (PLAY_BACK_PB — это командная кнопка **Обратное воспроизведение**):

```
case 206: /* Код [O] */
     /*-------------------------------------------------
     Пользователь нажал [O] — Обратное воспроизведение.
     -------------------------------------------------*/
     SendMessage ( hWnd,
                   WM_COMMAND,
                   PLAY_BACK_PB,
                   0);
     return 0L;
```

Одна и та же горячая клавиша [a] используется и для командной кнопки **Играть**, и для командной кнопки **Пауза**. Это позволяет воспроизводить и останавливать воспроизведение простым нажатием клавиши [a]. В случае нажатия [a] мы проверяем значение переменной giPlayEnable, которая показывает, когда воспроизводить, а когда делать паузу. Если мы

воспроизводим, то последующее нажатие клавиши [a] должно приводить к паузе; если сделали паузу, то нажатие клавиши [a] возобновляет воспроизведение:

```
case 192: /* Код русской буквы А */
     /*-------------------------------------------
     Пользователь нажал [A] для Играть, или для Паузы.
     -------------------------------------------*/
     if ( giPlayEnable == 1 )
        {
        SendMessage ( hWnd,
                      WM_COMMAND,
                      PAUSE_PB,
                      0);
        }
   else
        {
        SendMessage ( hWnd,
                      WM_COMMAND,
                      PLAY_PB,
                      0);
        }
   return 0L;
```

Вариант WM_HSCROLL программы Control2

В варианте WM_HSCROLL мы обрабатываем сообщения линеек прокрутки. В переменных giSpeedScroll и giPositionScroll хранятся значения текущего положения бегунков линеек прокрутки. Когда пользователь нажимает на кнопки со стрелками, кнопки **Медленнее** и **Быстрее** или перемещает бегунки линеек прокрутки, мы изменяем переменную giSpeedScroll (или переменную giPositionScroll) и выполняем функцию ChangePositionWasRequested(). Так, если пользователь нажимает кнопку линейки прокрутки со стрелкой влево, то выполняется фрагмент текста программы вариантов SB_PAGEUP и SB_LINEUP:

```
case SB_PAGEUP:
case SB_LINEUP:
    if ( GetDlgItem ( hWnd, SPEED_SB )
         == hWndCtrl )
        {
        /*--------------------------------
        Бегунок линейки прокрутки Скорость
        переместился влево.
        --------------------------------*/
        giSpeedScroll--;
        ChangeSpeedWasRequested ( hWnd );
        return 0L;
        }
     else
        {
        /*-------------------------------
        Бегунок линейки Перемотка/Вперед
        переместился влево.
        -------------------------------*/
        giPositionScroll-=2;
        ChangePositionWasRequested ( hWnd );
        return 0L;
        }
```

Обработка нажатий командных кнопок

Сообщения командных кнопок обрабатываются в варианте WM_COMMAND. Переменная giPlayEnable используется на протяжении всей "жизни" программы. Ее значение отображает текущее состояние процесса воспроизведения. Значение +1 показывает, что воспроизведение осуществляется в прямом направлении, –1 — что оно происходит в обратном направлении, а 0 — что воспроизведение отсутствует.

Когда пользователь нажимает командную кнопку **Играть**, выполняется фрагмент текста программы варианта PLAY_PB. В этом фрагменте изменяется значение переменной giPlayEnable, запрещается командная кнопка **Играть** и разрешаются командные кнопки **Обратное воспроизведение** и **Пауза** (таким образом, как только пользователь нажимает командную кнопку **Играть**, начинается воспроизведение, и командные кнопки **Пауза** и **Обратное воспроизведение** должны быть разрешены):

```
switch (wParam)
       {
       case PLAY_PB:
            /*-----------------------------------------
            Пользователь нажал кнопку Играть для начала
            воспроизведения.
            -----------------------------------------*/
            giPlayEnable = 1;

            /*-------------------
            Запрет кнопки Играть.
            -------------------*/
            hWndCtrl = GetDlgItem ( hWnd, PLAY_PB );
            EnableWindow ( hWndCtrl, 0 );

            /*-----------------------------------------
            Разрешение кнопки Обратное воспроизведение.
            -----------------------------------------*/
            hWndCtrl = GetDlgItem ( hWnd, PLAY_BACK_PB );
            EnableWindow ( hWndCtrl, 1 );

            /*-----------------------
            Разрешение кнопки Пауза.
            -----------------------*/
            hWndCtrl = GetDlgItem ( hWnd, PAUSE_PB );
            EnableWindow ( hWndCtrl, 1 );

            return 0L;
```

Аналогичным образом выполняем соответствующие действия при нажатии пользователем командных кнопок **Обратное воспроизведение** или **Пауза**.

Когда пользователь нажимает командные кнопки **Медленнее**, **Быстрее** или **Нормально**, изменяется значение переменной giSpeedScroll, а затем выполняется функция ChangeSpeedWasRequested(). Например, когда пользователь нажимает командную кнопку **Медленнее**, выполняется фрагмент текста программы варианта SLOW_PB:

```
case SLOW_PB:
     /*-----------------------------------
     Пользователь выбрал кнопку Медленнее.
     -----------------------------------*/
     giSpeedScroll--;
     ChangeSpeedWasRequested ( hWnd );
     return 0L;
```

Разрешение и запрет использования мыши

Пользователь может разрешить или запретить использовать мышь, выбрав команды меню **Запретить мышь** или **Разрешить мышь**. Соответствующими вариантами являются IDM_ENABLE_MOUSE и IDM_DISABLE_MOUSE (в варианте сообщения WM_COMMAND). Для разрешения использования мыши при воспроизведении используем функцию sp_EnableMouseDuringPlay(), а для запрета — функцию sp_DisableMouseDuringPlay():

```
case IDM_ENABLE_MOUSE:
     sp_EnableMouseDuringPlay();
     return 0;

     case IDM_DISABLE_MOUSE:
     sp_DisableMouseDuringPlay();
     return 0;
```

Глава 10. Автономные программы

Все приложения, которые мы разрабатывали в предыдущих главах, являлись неавтономными звуковыми программами. Это значит, что дистрибутивная дискета (дискета, содержащая все файлы программы) каждой из этих программ должна включать как исполняемый файл (.EXE) программы, так и звуковые файлы, используемые этой программой. Например, дистрибутивная дискета программы Dog должна содержать следующие файлы:

1. Dog.exe
2. Bark.ts
3. Hello.ts

Такая организация программ создает определенные неудобства при их распространении. Например, одно из неудобств состоит в том, что звуковые файлы должны быть записаны именно в той директории, которая указана в качестве первого параметра функции sp_OpenSession().

Чтобы устранить эти неудобства, вы можете преобразовать неавтономное приложение в автономное. Так, после преобразования программы Dog в автономную программу дистрибутивная дискета будет содержать только исполняемый файл, который включает и все необходимые программе звуковые файлы.

Преобразование программы Dog в автономную программу

Чтобы поупражняться, мы преобразуем сейчас неавтономную программу Dog (которая рассматривалась в предыдущих главах) в автономную программу, для чего выполним описанные ниже действия.

Шаг 1

Если вы используете компилятор Microsoft C, то в файл Dog.c нужно добавить такой оператор:

```
extern char **__argv;
```

Если вы используете компилятор Borland C, то добавьте в файл Dog.c следующий оператор:

```
extern char **_argv;
```

Этот оператор добавляется после операторов #include:

```
/*--------
 #include
 --------*/
/*-------------------------------------
 Требуется во всех приложениях Windows.
 -------------------------------------*/
#include <windows.h>

/*-------------------------------------------------------
 Необходимо для того, чтобы функции sp_, макросы SP_ и
 определения #define из библиотеки звуковой поддержки C
 можно было использовать в этих приложениях.
 -------------------------------------------------------*/
```

```
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-------------------------------------------------------
 Определения и прототипы, характерные для этого приложения.
 -------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\DogAlone.h"

/*-------------------------------------------------------
 Для стандартных функций C, используемых в этом приложении.
 -------------------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>

/*-------------------------------------------------
        *** Для компилятора Microsoft C ***

ПОСКОЛЬКУ ЭТО АВТОНОМНОЕ ПРИЛОЖЕНИЕ, НЕОБХОДИМ __argv.
-------------------------------------------------*/
extern char **__argv;

/*-------------------------------------------------
        *** Для компилятора Borland C ***

ПОСКОЛЬКУ ЭТО АВТОНОМНОЕ ПРИЛОЖЕНИЕ, НЕОБХОДИМ _argv.
-------------------------------------------------*/
extern char ** _argv;
```

Шаг 2

Если вы используете компилятор Microsoft C, то измените оператор в варианте WM_CREATE следующим образом:

```
case WM_CREATE:
     /*-----------------------------------
     Открываем звуковой сеанс лая собаки.
     -----------------------------------*/
     iOpenBarkResult = sp_OpenSession (
                       __argv[0],
                       SP_STAND_ALONE,
                       0L,
                       SP_TS_TYPE);
```

Для компилятора Borland C это будет выглядеть так:

```
case WM_CREATE:
     /*-----------------------------------
     Открываем звуковой сеанс лая собаки.
     -----------------------------------*/
     iOpenBarkResult = sp_OpenSession (
                       _argv[0],
                       SP_STAND_ALONE,
                       0L,
                       SP_TS_TYPE);
```

В приведенном вызове функции sp_OpenSession() мы использовали __argv[0] (или _argv[0]) в качестве первого параметра и SP_STAND_ALONE — в качестве второго параметра.

Вот и все изменения, которые нужно сделать в исходном тексте программы для преобразования его в автономную программу.

Шаг 3

На этом шаге мы компилируем и компонуем автономную программу Dog.

Для компиляции и компоновки компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
NMAKE Dog.mak    [Enter]
```

Для компиляции и компоновки компилятором фирмы Borland нужно выполнить такие действия:

- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
MAKE -f Dog.bmk    [Enter]
```

Шаг 4

Последний шаг — это объединение файла Dog.exe (сгенерированного на шаге 3) и звукового файла Bark.ts, расположенного в директории c:\spSDK\SFiles\.

Программа Dog использует два звуковых файла — Bark.ts и Hello.ts. Для объединения этих файлов с исполняемым файлом необходим редактор Sound Editor фирмы TS. Программа Sound Editor позволяет включить в исполняемый файл неограниченное количество звуковых файлов. Последние могут быть типа S, TS, WAV, VOC или SND.

Поскольку мы предположили, что у вас нет редактора Sound Editor фирмы TS, то будем использовать утилиту TSlink.exe. Эта утилита позволяет подключить только один звуковой файл (типа S или TS).

Утилита TSlink

Утилита TSlink расположена в директории c:\spSDK\Util\. Эта утилита позволяет объединять исполняемый файл с отдельным звуковым файлом (последний файл может быть типа S или TS).

Для вызова TSlink выполните следующие действия:

- Войдите в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS введите:

```
c:\spSDK\Util\TSlink Dog.exe c:\spSDK\TSFiles\Bark.ts DogAlone.exe
```

Первый параметр утилиты TSlink — это имя исполняемого файла, к которому будет присоединен звуковой файл; второй параметр — имя присоединяемого звукового файла; третий параметр — имя результирующего исполняемого файла автономной программы. Расширения имен файлов нужно указывать обязательно: в качестве первого параметра должно быть указано имя Dog.exe, а не Dog, в качестве второго — Bark.ts, а не Bark. Соответственно третьим параметром должно быть имя DogAlone.exe, а не DogAlone.

Примечание:

Утилита TSlink.exe объединяет исполняемый файл со звуковым файлом.

Синтаксис: TSlink первый_параметр второй_параметр третий_параметр

Первый параметр: Имя исполняемого файла, к которому будет подключаться звуковой файл.

Второй параметр: Имя звукового файла, который будет подключаться.

Третий параметр: Имя результирующего исполняемого файла.

Выполнение автономной программы Dog

Теперь можно выполнять автономную программу DogAlone.

Для выполнения данной программы из Windows нужно сделать следующее:

- Выберите из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- В диалоговом окне **Пролистать** выберите файл .

```
c:\spSDK\Samp4Win\DogAlone.exe
```

Для выполнения программы из DOS:

- Завершите Windows.
- Войдите в директорию c:\spSDK\Samp4Win\.
- После приглашения DOS введите:

```
WIN DogAlone    [Enter]
```

Автономная программа состоит из одного файла — DogAlone.exe. Этот файл может находиться на любом устройстве и в любой директории.

С помощью утилиты TSlink мы произвели объединение звукового файла Bark.ts с файлом Dog.exe, в результате чего был создан новый файл с именем DogAlone.exe. Объем файла DogAlone.exe приблизительно равен сумме объемов файлов Dog.exe и Bark.ts. Теперь дистрибутивная дискета будет содержать только один файл — DogAlone.exe. Таким образом, отпадает необходимость в распространении звукового файла Bark.ts.

Применение утилиты TSLabels

Сокращенная версия утилиты TSlink позволяет подсоединять только один звуковой файл. Следовательно, нельзя подключить к программе Dog еще и файл Hello.ts. Однако если у вас есть редактор Sound Editor фирмы TS, то вы можете создать отдельный звуковой файл, объединяющий файл Bark.ts и звуковой фрагмент "Good-Bye" файла Hello.ts. Можно выделить звуковой фрагмент "Good-Bye" метками, в начале и конце фрагмента, например: "GoodBye начинается здесь" и "GoodBye заканчивается здесь". В этом случае для воспроизведения фрагмента Good-Bye применяется следующий оператор:

```
sp_PlayLabelF ( "GoodBye начинается здесь",
                "GoodBye заканчивается здесь");
```

Функция sp_PlayLabelF() не включена в сокращенную версию звуковой библиотеки функций С фирмы TS, которая поставляется на дискете вместе с данной книгой.

Параметр __argv[]

В автономном приложении переменная _ _argv[] выполняет те же функции, что и argv[] в DOS-программах на С. Она содержит параметры, которые задаются при вызове программы. В автономном приложении при вызове функции sp_OpenSession() в качестве первого параметра должно быть указано _ _argv[0], т.к. этот параметр содержит путь и имя выполняемой программы. Например, если вы выполняете программу DogAlone.exe, расположенную в директории c:\spSDK\Samp4Win\, то значение _ _argv[0] равно c:\spSDK\Samp4Win\DogAlone.exe.

Как было отмечено ранее, при использовании компилятора Microsoft C нужно применять __argv, а при использовании компилятора Borland C — _argv.

Третий параметр функции sp_OpenSession()

Третий параметр функции sp_OpenSession(), содержащейся в сокращенной версии звуковой библиотеки функции C фирмы TS, не используется, поэтому вместо него можно задавать 0L.

Преобразование других программ

Таким же способом можно преобразовать в автономную любую из приведенных в этой книге неавтономных программ.

Программа WhoAmI

Последняя программа в данной главе — WhoAmI. Эта программа отображает на экране свое имя и путь, а затем сама воспроизводит звуковой файл.

Когда пользователь выбирает из меню программы команду **Играть**, появляется окно сообщения, содержащее значение argv[0], а затем воспроизводится подключенный звуковой файл.

Ниже приведен полный листинг программы WhoAmI.

Листинг 10.1. WhoAmI.h

```
/*====================================
 ИМЯ ФАЙЛА: WhoAmI.h

 (C) Copyright Gurewich 1992, 1993
 ====================================*/

/*---------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc      ( HWND, UINT, UINT, LONG ) ;
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG ) ;

/*-------
 #define
 -------*/
#define IDM_QUIT          1    /* Команда меню Завершить.   */
#define IDM_ABOUT         2    /* Команда меню О Программе.*/
#define IDM_PLAY          3    /* Команда меню Играть.      */


/*--------------------
 Глобальные переменные
 --------------------*/
char       gszAppName[] = "WhoAmI" ; /* Имя нашего приложения. */
HINSTANCE  ghInst;                   /* Текущий экземпляр.     */
```

Листинг 10.2. WhoAmI.c

```
/*=========================================================================
 ПРОГРАММА: WhoAmI.c
 ---------
 (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

 Как компилировать, компоновать и создавать эту автономную программу:
 =========================================================================

   1. Если вы используете компилятор Microsoft C, убедитесь в том, что
      в этом файле используется __argv (а не _argv)
      [] Удостоверьтесь, что ваш ПК находится в защищенном режиме DOS.
      [] Войдите в директорию c:\spSDK\Samp4Win\
      [] После приглашения DOS введите:

                NMAKE  WhoAmI.mak    [Enter]

   2. Если вы используете компилятор Borland C, убедитесь в том, что
      в этом файле используется _argv (а не __argv)
      [] Войдите в директорию c:\spSDK\Samp4Win\
      [] После приглашения DOS введите:

                MAKE -f WhoAmI.bmk    [Enter]

   3. После приглашения DOS введите:

      c:\spSDK\Util\Tslink WhoAmI.exe c:\spSDK\TSfiles\Hello.ts
      Who.exe    [Enter]

 Сейчас у вас есть приложение Who.exe, которое является автономным.

=========================================================================*

/*-------
 #include
 --------*/
/*-----------------------------------------------
 windows.h требуется во всех приложениях Windows.
 -----------------------------------------------*/
#include <windows.h>

/*-------------------------------------------------------------
 Требуется для того, чтобы в этом приложении могли быть
 использованы функции sp_, а также макросы SP_ и определения
 #define из библиотеки звуковой поддержки C.
 -------------------------------------------------------------*/
#include "c:\spSDK\TegoWlib\sp4Win.h"

/*-----------------------------------------------------
 Определения и прототипы, характерные для этого приложения.
 -----------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\WhoAmI.h"

/*-------------------------------------------------
 Для функций C,требующих стандартные #include-файлы C.
 -------------------------------------------------*/
```

```
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>

/*=========== Для компилятора Microsoft C ================*/
/*** extern char ** __argv;***/

/*=========== Для компилятора Borland C ==================*/
extern char ** _argv;

/*================
 ФУНКЦИЯ: WinMain()
 ================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*--------------------------------
 Локальные и статические переменные.
 --------------------------------*/
HWND       hWnd; /* Идентификатор окна нашего приложения.    */
MSG        msg;  /* Сообщение, которое будет обрабатываться */
                 /* нашим приложением.                       */
WNDCLASS   wc;   /* Класс окна нашего приложения.            */

/*--------------------------------------
 Сделаем переменную hInstance глобальной.
 --------------------------------------*/
ghInst = hInstance;

/*------------------------------------------------------------
 Обновляем структуру класса окна и регистрируем класс окна.
 ------------------------------------------------------------*/
if ( !hPrevInstance )
   {
   /*---------------------------------------
   Условие if удовлетворено, это самый первый
   запуск данного приложения.
   ---------------------------------------*/
   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;
   wc.cbWndExtra    = 0 ;
   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfTape" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW      ) ;
   wc.hbrBackground = GetStockObject ( WHITE_BRUSH );
   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = gszAppName ;

   /*----------------
   Регистрируем окно.
   ----------------*/
   RegisterClass ( &wc );
   }/* конец условия if( !hPrevInstance ) */
```

```
/*-------------------------------
 Создаем окно нашего приложения.
 -------------------------------*/
hWnd = CreateWindow ( gszAppName,
                      gszAppName,
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      CW_USEDEFAULT,
                      NULL,
                      NULL,
                      ghInst,
                      NULL );

/*---------------------------------------------
 Отображаем и обновляем окно нашего приложения.
 ---------------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );

/*--------------
 Цикл сообщений
 --------------*/
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage  ( &msg );
      }

return msg.wParam ;

} /* Конец функции. */
/*========================= конец WinMain() ====================*/


/*==================
 ФУНКЦИЯ: WndProc()
 ==================*/
/*-------------------------------
 НАЗНАЧЕНИЕ: Обработка сообщений.
 -------------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
static FARPROC lpfnAboutDlgProc;/* Для диалогового окна О Программе. */

static int     iOpenResult;
```

```
switch ( message )
        {
        case WM_CREATE:
             /*-----------------------------
             Открываем сеанс работы со звуком.
             -----------------------------*/
             iOpenResult = sp_OpenSession (
                           _argv[0],
                           SP_STAND_ALONE,
                           0L,
                           SP_TS_TYPE);

             /*-------------------------------------------
             Завершаем работу, если невозможно открыть файл.
             -------------------------------------------*/
             if ( iOpenResult != SP_NO_ERRORS )
                {
                MessageBox ( NULL,
                             "Невозможно открыть звуковой сеанс!",
                             "Сообщение от WhoAmI.c",
                             MB_ICONINFORMATION );

                /*--------------------
                Завершаем приложение.
                --------------------*/
                SendMessage ( hWnd,
                              WM_DESTROY,
                              0,
                              0 );
                }

             /*-----------------------------------------
             Получаем lpfnAboutDlgProc диалогового окна
             О Программе.
             -----------------------------------------*/
             lpfnAboutDlgProc =
             MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);
             return 0;

        case WM_COMMAND:
             /*-------------------------
             Обрабатываем команды меню.
             -------------------------*/
             switch (wParam)
                    {
                    case IDM_QUIT:
                         /*-----------------------------------
                         Пользователь выбрал команду Завершить.
                         -----------------------------------*/
                         DestroyWindow (hWnd);
                         return 0L;

                    case IDM_ABOUT :
                         /*-------------------------------------
                         Пользователь выбрал команду О Программе.
                         -------------------------------------*/
```

```
                    DialogBox ( ghInst,
                                "AboutBox",
                                hWnd,
                                lpfnAboutDlgProc );
                    return 0L;

                case IDM_PLAY :
                    /*--------------------------------
                    Пользователь выбрал команду Играть.
                    --------------------------------*/
                    MessageBox ( NULL,
                                 _argv[0],
                                 "Значение argv[0] :",
                                 MB_ICONINFORMATION );

                    sp_PlayF ( SP_START_OF_FILE,
                               SP_END_OF_FILE ) ;

                    return 0L;

                }/* конец оператора switch (wParam) */

        case WM_DESTROY:
                PostQuitMessage (0);
                return 0;
        }/* конец оператора switch (message) */

/*-----------------------
 Сообщение не обработано.
 -----------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}/* конец WndProc() */
/*=================== конец WndProc() ====================*/

/*=======================
 ФУНКЦИЯ: AboutDlgProc()
 =======================*/
/*----------------------------------------
 НАЗНАЧЕНИЕ:
 Это процедура диалогового окна О Программе.
 ----------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
        {
        case WM_INITDIALOG :
                return TRUE;

        case WM_COMMAND :
                switch ( wParam )
                        {
                        case IDOK :
```

```
                    case IDCANCEL :
                         EndDialog ( hDlg, 0 );
                         return TRUE;
                    }
        }/* конец switch(message) */
return FALSE ;
}/* Конец функции. */
/*=============== Конец AboutDlgProc() ==============*/
```

Листинг 10.3. WhoAmI.rc

```
/*=========================================
 ИМЯ ФАЙЛА: WhoAmI.rc
 ----------

 ОПИСАНИЕ ФАЙЛА:
 ---------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 =========================================*/

/*-------
 #include
 --------*/
#include <windows.h>
#include "WhoAmI.h"

/*---
 Меню
 ----*/
WhoAmI  MENU
BEGIN
   POPUP  "&Меню"
      BEGIN
          MENUITEM "&Завершить",        IDM_QUIT
          MENUITEM "&О Программе",      IDM_ABOUT
          MENUITEM "&Играть",           IDM_PLAY
      END
END

/*-------------------------------
 Определение пиктограммы кассеты.
 Имя файла: Tape.ico
 Имя пиктограммы: IconOfTape
 -------------------------------*/
IconOfTape ICON Tape.ico

/*---------------------------
 Диалоговое окно О Программе.
 ---------------------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "О программе WhoAmI"
FONT 8, "MS Sans Serif"
```

```
BEGIN
    PUSHBUTTON          "OK", IDOK, 64, 75, 40, 14
    CTEXT               "(C) Copyright Gurewich 1992, 1993",
                        -1, 13, 47, 137, 18
    ICON                "IconOfTape", -1, 14, 12, 18, 20
END
```

Листинг 10.4. WhoAmI.def

```
;======================================
; Файл определения модуля для WhoAmI.c
;======================================

NAME            WhoAmI

DESCRIPTION     'Программа WhoAmI. (C) Copyright Gurewich 1992, 1993'

EXETYPE         WINDOWS

STUB            'WINSTUB.EXE'

CODE  PRELOAD  MOVEABLE DISCARDABLE

DATA  PRELOAD  MOVEABLE MULTIPLE

HEAPSIZE       1024
STACKSIZE      8192
```

Листинг 10.5. WhoAmI.mak

```
#============
# WhoAmI.mak
#============

WhoAmI.exe : WhoAmI.obj WhoAmI.h WhoAmI.def WhoAmI.res
    link /nod WhoAmI.obj, WhoAmI.exe, NUL, \
        slibcew.lib oldnames.lib libw.lib commdlg \
        c:\spSDK\TegoWlib\TegoWin.lib, \
        WhoAmI.def
    rc -t WhoAmI.res

WhoAmI.obj : WhoAmI.c WhoAmI.h
   cl -c -G2sw -Ow -W3 -Zp  WhoAmI.c

WhoAmI.res : WhoAmI.rc WhoAmI.h Tape.ico
   rc -r WhoAmI.rc
```

Компиляция и компоновка программы WhoAmI

Для компиляции и компоновки программы WhoAmI компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win.

➤ После приглашения DOS введите:

```
NMAKE WhoAmI.mak    [Enter]
```

Для компиляции и компоновки программы WhoAmI компилятором фирмы Borland выполните следующие действия:

➤ Войдите в директорию c:\spSDK\Samp4Win.

➤ После приглашения DOS введите:

```
MAKE -f WhoAmI.bmk    [Enter]
```

Объединение программы WhoAmI со звуковым файлом

При вызове функции sp_OpenSession() программы WhoAmI используется файл типа TS:

```
iOpenResult = sp_OpenSession (
                  __argv[0],
                  SP_STAND_ALONE,
                  0L,
                  SP_TS_TYPE);
```

Первый параметр sp_OpenSession() указывает на то, что звуковой файл, который следует открыть, — это сама программа, второй параметр — на то, что программа является автономной. Третий параметр sp_OpenSession() не используется в сокращенной версии библиотеки TS, так что мы задаем значение 0L. Четвертый параметр sp_OpenSession() указывает на то, что программа должна быть объединена со звуковым файлом типа TS.

Для подсоединения звукового файла типа TS Hello.ts к исполняемому файлу WhoAmI.exe введите после приглашения DOS

```
c:\spSDK\Util\Tslink WhoAmI.exe c:\spSDK\TSfiles\Hello.ts Who.exe [Enter]
```

Результирующий файл Who.exe является автономной программой.

Переменная lpszCmdLine

Как мы знаем, при вызове функции WinMain() в качестве третьего параметра ей передается lpszCmdLine.

```
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
.................
.................
.................
}
```

Переменная lpszCmdLine содержит параметры, которые были введены при запуске приложения Windows. Предположим, что приложение Windows называется PlayAny.exe. Для выполнения программы PlayAny из DOS пользователь должен ввести

```
WIN PlayAny Имя_звукового_файла    [Enter]
```

Если пользователь выполняет программу PlayAny из Windows, выбирая из меню **Файл** команду **Выполнить**, то нужно ввести:

```
PlayAny Имя_звукового_файла
```

В таких случаях для того, чтобы получить имя звукового файла, который будет воспроизводиться, можно воспользоваться третьим параметром функции WinMain().

Но для автономных приложений файлом, который будет воспроизводиться, является само приложение, и поскольку переменная lpszCmdLine имя приложения не содержит, упомянутый выше способ получения имени звукового файла применить нельзя. По этой причине мы и используем argv[0].

Глава 11. Использование звуковой платы в приложениях Windows

Пользователи, в компьютерах которых установлена звуковая плата, имеют возможность наслаждаться более хорошим качеством звука, чем те, кто использует встроенный динамик. Таким образом, при создании Windows-приложений целесообразно предусмотреть в программе возможность работы с обоими (с точки зрения звукового сопровождения) типами систем ПК — со звуковой платой и без нее. Хорошо спроектированная программа должна определять наличие или отсутствие в системе звуковой платы и соответственно выбирать устройство для воспроизведения звука — встроенный динамик компьютера либо звуковую плату.

Между воспроизведением через встроенный динамик и с помощью звуковой платы есть два основных различия. Во-первых, звуковая плата позволяет воспроизводить звук более качественно. Во-вторых, она не использует ресурсы процессора. Это означает, что после того как процессор направит звуковой плате команду воспроизвести звуковой фрагмент, она выполнит эту команду самостоятельно, без каких-либо дополнительных действий со стороны процессора ПК. Таким образом, в ходе воспроизведения процессором остается свободным и может выполнять другие задачи, например перемещать графические объекты.

Оболочка Windows и звуковые платы

Одним из главных преимуществ оболочки Windows является то, что при программировании в среде Windows пользователь имеет в своем распоряжении огромное количество программ, уже инсталлированных на его ПК.

Например, когда текст или графика отображаются на мониторе, ваша программа не должна "беспокоиться" о типе этого монитора, поскольку ответственность за отображение текста или графики на мониторе пользователя возложена на Windows. Как программист, вы можете с уверенностью предположить, что пользователь уже инсталлировал соответствующий драйвер монитора в своей системе и что оболочку Windows этот драйвер устраивает. Если драйвер дисплея несовместим с этой оболочкой, то у пользователя возникнут проблемы при работе с Windows. Ваша программа предполагает, что компьютер уже сконфигурирован и эта конфигурация принята Windows.

То же, естественно, относится и к принтеру: работа вашей программы строится на предположении, что принтер, независимо от его типа, хорошо работает с оболочкой Windows. Эта концепция известна как свойство *аппаратной независимости* Windows.

Свойство аппаратной независимости распространяется и на звуковые платы. Это означает, что если установленная звуковая плата совместима с Windows, то все программы, представленные в данной и последующих главах, будут хорошо с этой платой работать.

Инсталляция звуковой платы

Инсталляция звуковой платы на вашем ПК включает два отдельных шага:

1. Аппаратная и программная инсталляция.
2. Установка драйверов Windows.

Аппаратная и программная инсталляция звуковой платы

При инсталляции звуковой платы на компьютере следуйте руководству по инсталляции, поставляемому в комплекте со звуковой платой. В руководстве поясняется, как установить плату в один из разъемов ПК и как инсталлировать поставляемое в комплекте с ней программное обеспечение.

В компьютере, который использовали авторы этой книги, была установлена звуковая плата Sound Blaster Pro фирмы Creative Labs. Когда инсталляция платы была завершена, мы запустили тестовую утилиту TEST-SBP.EXE, которая поставлялась вместе со звуковой платой. Утилита TEST-SBP.EXE (расположенная в директории C:\SBPRO\) проверила, как проведена инсталяция, и отобразила на экране компьютера различные установки платы: адрес ввода/вывода, номер прерывания и номер канала прямого доступа к памяти ПДП (DMA).

Если вы тоже используете звуковую плату Sound Blaster Pro, то и вам потребуется знать эти установки для инсталляции драйвера Windows, так что проверьте, записали ли вы адрес ввода/вывода, номер прерывания и номер канала ПДП, которые были выведены на экран утилитой TEST-SBP.EXE.

Если у вас другая звуковая плата, то вы должны будете получить аппаратные установки этой платы, используя поставляемые в комплекте с ней эквивалентные программные утилиты.

Драйверы Windows для звуковой платы

Для применения звуковой платы в приложениях Windows необходимо установить драйверы звуковой платы (если они еще не установлены).

Проверка и верификация текущих установленных драйверов

Для проверки установленных в вашей системе драйверов выполните следующие действия:

- Выберите в окне группы программ **Главная** пиктограмму**Панель Управления** (см. рис. 11.1). Появится окно **Панель Управления**, изображенное на рис. 11.2.
- В окне группы программ **Панель Управления** выберите пиктограмму **Драйверы**. Появится окно **Драйверы**, изображенное на рис. 11.3. Как видно из рисунка, это окно содержит список **Установленные Драйверы** (т.е. драйверы, которые установлены в системе на момент отображения окна).

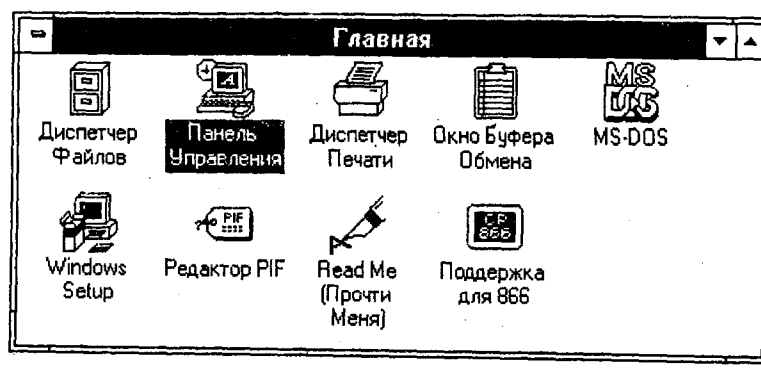


Рис. 11.1. Пиктограмма Панель Управления

Рис. 11.2. Окно Панель Управления

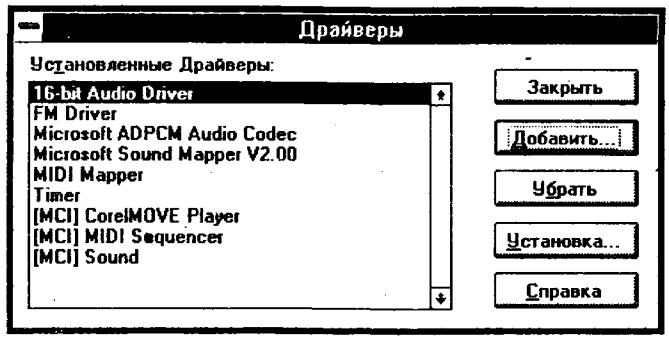


Рис. 11.3. Окно Драйверы

Как можно увидеть, драйверы, которые мы хотели установить (драйверы звуковой платы) уже включены в список установленных драйверов. В программах, описанных в этой и последующих главах, мы будем пользоваться некоторыми из драйверов, которые вы видите в списке на рис. 11.3.

Драйверы

При работе со звуковой платой Sound Blaster Pro потребуются три драйвера:

- Creative Sound Blaster Pro Auxiliary Audio
- Creative Sound Blaster Pro MIDI Synthesizer
- Creative Sound Blaster Pro Wave and MIDI

Если у вас другая звуковая плата фирмы Creative Labs (или другого производителя), то нужно установить драйверы, поставляемые в комплекте с вашей платой (*при переводе данной книги использовалась звуковая плата из комплекта Sound Kit (2) фирмы Peacock).*

Кроме драйверов звуковой платы в окне, изображенном на рис. 11.3, можно увидеть и другие установленные драйверы:

- MIDI Mapper
- Timer
- [MCI] MIDI Sequencer
- [MCI] Sound

Эти драйверы поставляются в комплекте с оболочкой Windows. Они необходимы для работы с мультимедиа.

Установка драйверов

Если какой-либо из драйверов, представленных на рис. 11.3, в вашей системе не установлен, то его можно добавить. Для этого нужно:

➤ Выбрать в окне **Драйверы** командную кнопку **Добавить**.

Появится окно **Добавить**, показанное на рис. 11.4. Это окно содержит список драйверов, которые вы можете добавить.

Например, если вам нужно установить драйвер [MCI] CD Audio, выберите его (щелкните на его имени кнопкой мыши — имя будет выделено), а затем нажмите командную кнопку **OK**.

Если выбранный драйвер не появился в списке, то выберите в этом же списке элемент **Неуказанный или Обновленный драйвер**. Затем выполните щелчок на командной кнопке **OK** и следуйте дальнейшим инструкциям оболочки Windows. Windows попросит вас указать устройство и директорию драйвера, который вы хотите установить.

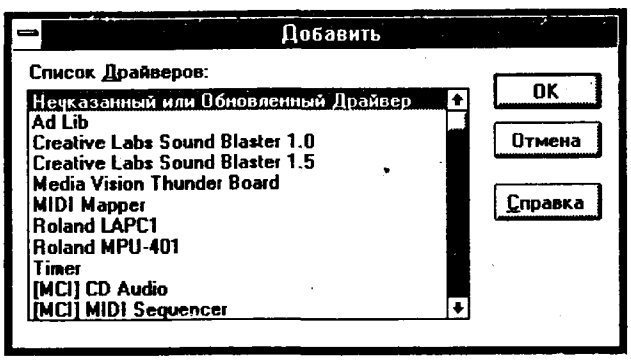


Рис. 11.4. Окно Добавить

Примечание:

После завершения установки драйвера Windows предложит перезапустить систему. Имейте в виду, что вновь установленный драйвер начнет действовать только после ее перезапуска.

Программа Hello2

Сейчас мы напишем наше первое приложение Windows для звуковой платы — программу Hello2.

Запись файла .WAV для программы Hello2

Программа Hello2 воспроизводит файл c:\spSDK\WAV\Hello2.wav. Ниже описан процесс создания файла типа WAV.

➤ Выберите пиктограмму **Звукозапись** из группы программ Реквизиты (см. рис. 11.5).

➤ Появится окно **Звукозапись**, изображенное на рис. 11.6. Программа Звукозапись поставляется в комплекте с оболочкой Windows. Эта программа позволяет записывать файлы с расширением .WAV и воспроизводить их, используя звуковую плату. Несколько позже вы научитесь самостоятельно создавать программы, подобные программе Звукозапись. А пока продолжим запись звукового файла для программы Hello2. Убедитесь в том, что микрофон правильно подключен к звуковой плате, и приготовьтесь кое-что записать.

В окне **Звукозапись** выберите командную кнопку **Микрофон** (крайняя справа кнопка, на которой изображен микрофон).

Теперь вы должны нажать эту кнопку и произнести в микрофон такую фразу:

Привет. Это моя первая звуковая программа для Windows!

Рис. 11.5. Пиктограмма Звукозапись

Рис. 11.6. Окно Звукозапись

- Для остановки записи воспользуйтесь командной кнопкой **Стоп** (она расположена слева от кнопки **Микрофон**).

Чтобы прослушать то, что вы записали:

- Нажмите командную кнопку **Воспроизведение** (эта кнопка расположена слева от кнопки **Стоп**).

Если вы удовлетворены качеством записи, то сохраните звуковой файл, выбрав из меню **Файл** команду **Сохранить Как**. Сохраните вашу запись как c:\spSDK\WAV\Hello2.wav. Если качество записи вас не удовлетворяет, то выберите из меню **Файл** команду **Новый** и повторяйте весь процесс записи до тех пор, пока не получите подходящее качество звучания.

Компиляция, компоновка и выполнение программы Hello2

Для того чтобы лучше разобраться в программе Hello2, рекомендуем вам откомпилировать, скомпоновать и выполнить эту программу прежде чем начать изучать ее текст.

Для компиляции и компоновки программы компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.

- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
NMAKE Hello2.mak    [Enter]
```

Для компиляции и компоновки программы компилятором фирмы Borland:

- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
MAKE -f Hello2.bmk    [Enter]
```

Перед выполнением программы Hello2 убедитесь в том, что на вашем ПК установлены Windows-совместимая звуковая плата и все необходимые драйверы и что файл Hello2.wav находится в директории c:\spSDK\WAV\.

Для выполнения программы Hello2:

- Выберите из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- В диалоговом окне **Пролистать** выберите файл

```
c:\spSDK\Samp4Win\Hello2.exe
```

Появится главное окно программы Hello2, приведенное на рис. 11.7.

Рис. 11.7. Главное окно программы Hello2

Меню программы Hell2 содержит две команды — **Завершить** и **О Программе** (см. рис. 11.8).

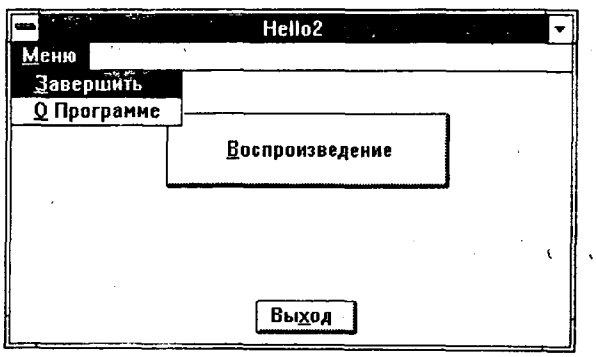


Рис. 11.8. Меню программы Hello2

Если пользователь выбирает команду **Завершить**, то программа завершается. При выборе команды **О Программе** появляется диалоговое окно команды **О Программе**, изображенное на рис. 11.9.

Рис. 11.9. Диалоговое окно команды О Программе программы Hello2

Когда пользователь нажимает командную кнопку **Воспроизведение**, звуковой файл типа WAV c:\spSDK\WAV\Hello2.wav воспроизводится с помощью звуковой платы. Во время воспроизведения разрешено использовать мышь, и пользователь может переключаться в другие приложения Windows.

Файлы программы Hello2

Программа Hello2 состоит из следующих файлов:

Hello2.h	#include-файл программы
Hello2.c	Текст программы на C
Hello2.rc	Файл ресурсов программы
Hello2.def	Файл определения модуля программы
Hello2.mak	Make-файл программы
ExtSpkr.ico	Пиктограмма программы

Эти файлы доступны в директории c:\spSDK\Samp4Win\. Все файлы Hello2 приведены в листингах 11.1–11.5.

Листинг 11.1. Hello2.h

```
/*=================================================
 ИМЯ ФАЙЛА: Hello2.h

 (C) Copyright Gurewich 1992, 1993
 =================================================*/

/*---------
 прототипы
 ---------*/
long FAR PASCAL _export WndProc       ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL _export AboutDlgProc ( HWND, UINT, UINT, LONG ) ;
```

```
/*------
 #define
 -------*/
#define IDM_QUIT   1   /* Команда меню Завершить    */
#define IDM_ABOUT  2   /* Команда меню О Программе */

#define PLAY_PB   100  /* Кнопка Воспроизведение. */
#define EXIT_PB   101  /* Кнопка Выход. */

/*--------------------
 Глобальные переменные
 --------------------*/
char       gszAppName[] = "Hello2" ; /* Имя нашего приложения. */
HINSTANCE  ghInst;                   /* Текущий экземпляр.     */
```

Листинг 11.2. Hello2.c

```
/*=============================================================
  ПРОГРАММА: Hello2.c
  ---------
  (C) Copyright 1992, 1993 Gurewich. (R) All rights reserved.

  ОПИСАНИЕ ПРОГРАММЫ:
  ------------------
  Эта программа типа Generic1.

  Программа воспроизводит файл типа WAV через звуковую плату.
 =============================================================*/

/*-------
 #include
 --------*/
/*-----------------------------------------------
 windows.h требуется во всех приложениях Windows.
 -----------------------------------------------*/
#include <windows.h>

/*-------------------------------------------------
 Требуется для того, чтобы функции ts_, макросы TS_ и
 определения #define из библиотеки звуковой поддержки
 могли быть использованы в этом приложении.
 -------------------------------------------------*/
#include "c:\spSDK\TegoWlib\ts4Win.h"

/*-------------------------------------------------------
 Определения и прототипы, характерные для этого приложения.
 -------------------------------------------------------*/
#include "c:\spSDK\Samp4WIN\Hello2.h"

/*-------------------------------------------------
 Для функций C, требующих стандартные #include-файлы C.
 -------------------------------------------------*/
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
```

```
/*=================
 ФУНКЦИЯ: WinMain()
 ==================*/
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR  lpszCmdLine,
                     int    nCmdShow )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
HWND      hWnd;   /* Указатель на окно нашего приложения.     */
MSG       msg;    /* Сообщение, которое будет обрабатываться */
                  /* нашим приложением.                       */
WNDCLASS  wc;     /* Класс окна нашего приложения.            */

HBITMAP   hBitmapOfBkGnd; /* Растровое изображение фона программы. */
HBRUSH    hBrushOfBkGnd;  /* Кисть фона программы. */

/*--------------------------------------
 Создадим глобальную переменную hInstance.
 --------------------------------------*/
ghInst = hInstance;

/*-------------------------------------------------------------
 Загрузим растровое изображение фона программы и создадим кисть.
 Фон определен в файле .rc и базируется на файле BACK2.bmp.
 -------------------------------------------------------------*/
hBitmapOfBkGnd = LoadBitmap ( ghInst, "BackGround" );
hBrushOfBkGnd = CreatePatternBrush ( hBitmapOfBkGnd );

/*-------------------------------------------------------------
 Обновляем структуру класса окна и регистрируем класс окна.
 -------------------------------------------------------------*/
if ( !hPrevInstance )
   {
   /*-----------------------------------------
   Условие if удовлетворяется; это самый первый
   запуск приложения.
   -----------------------------------------*/

   wc.style         = CS_HREDRAW | CS_VREDRAW ;
   wc.lpfnWndProc   = WndProc ;
   wc.cbClsExtra    = 0 ;

   wc.cbWndExtra    = DLGWINDOWEXTRA ;  /* Диалоговое окно как */
                                        /* главное окно.       */

   wc.hInstance     = ghInst ;
   wc.hIcon         = LoadIcon   ( ghInst, "IconOfExtSpkr" ) ;
   wc.hCursor       = LoadCursor ( NULL, IDC_ARROW         ) ;

   wc.hbrBackground = hBrushOfBkGnd; /* Кисть, которую мы создали с*/
                                     /* помощью растрового изображения.*/

   wc.lpszMenuName  = gszAppName ;
   wc.lpszClassName = "Hello2Class" ; /* Как указано в CLASS файла .RC.*/
```

```
    /*-----------------
     Регистрируем окно.
     -----------------*/
    RegisterClass ( &wc );

    }/* конец условия if( !hPrevInstance ) */

/*-------------------------------------------------------------------
ПРИМЕЧАНИЕ: В этом приложении главное окно — диалоговое.
            "Hello2Box" — это имя диалогового окна, как
            определено в файле .RC (перед ключевым словом DIALOG).
-------------------------------------------------------------------*/
hWnd = CreateDialog ( ghInst, "Hello2Box", 0, NULL );

/*-----------------------------------------
 Отображаем и обновляем окно нашего приложения.
 -----------------------------------------*/
ShowWindow   ( hWnd, nCmdShow );
UpdateWindow ( hWnd );

/*-------------
 Цикл сообщений
 -------------*/
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage  ( &msg );
      }

/*--------------------
 Удаляем объекты фона.
 --------------------*/
DeleteObject ( hBitmapOfBkGnd );
DeleteObject ( hBrushOfBkGnd );

return msg.wParam ;

} /* Конец функции. */
/*========================= конец WinMain() ====================*/

/*=================
 ФУНКЦИЯ: WndProc()
 =================*/
/*-------------------------------
 НАЗНАЧЕНИЕ: Обработка сообщений.
 -------------------------------*/
long FAR PASCAL _export WndProc ( HWND hWnd,
                                  UINT message,
                                  UINT wParam,
                                  LONG lParam )
{
/*---------------------------------
 Локальные и статические переменные.
 ---------------------------------*/
int    iOpenResult;  /* Результат открытия сеанса. */
int    iPlayResult;  /* Результат воспроизведения. */
```

```
char  sWAVFileName[125]; /* Имя файла .WAV для воспроизведения. */
char  sMessage[125];

/*-----------------------------------------
 Необходимо для диалогового окна О Программе.
 -----------------------------------------*/
static FARPROC lpfnAboutDlgProc ;

switch ( message )
       {
       case WM_CREATE:
            /*-------------------------------------------------------
            Получаем lpfnAboutDlgProc диалогового окна О Программе.
            -------------------------------------------------------*/
            lpfnAboutDlgProc =
            MakeProcInstance (( FARPROC) AboutDlgProc, ghInst);

            /*----------------------------------------
            Существует ли в системе звуковая плата,
            необходимая для воспроизведения файлов .WAV ?
            ----------------------------------------*/
          , if ( ts_WaveDeviceCanPlay() == 0 )
               {
               MessageBox ( NULL,
                            "Звуковая плата не обнаружена!",
                            "Сообщение",
                            MB_ICONSTOP );
               /*-------------------
               Завершаем приложение.
               -------------------*/
               SendMessage ( hWnd,
                             WM_DESTROY,
                             0,
                             0 );
               return 0;
               }

            /*-----------------------
            Открываем звуковой сеанс.
            -----------------------*/
            lstrcpy( sWAVFileName, "c:\\spSDK\\WAV\\Hello2.wav");
            iOpenResult =
            ts_OpenWaveSession ( sWAVFileName );

            if ( iOpenResult != 1 )
               {
               lstrcpy ( sMessage, "Невозможно открыть сеанс для :");
               lstrcat ( sMessage, sWAVFileName );
               MessageBox ( NULL,
                            sMessage,
                            "Сообщение",
                            MB_ICONINFORMATION );

               /*-------------------
               Завершаем приложение.
               -------------------*/
```

```
            SendMessage ( hWnd,
                          WM_DESTROY,
                          0,
                          0 );

            return 0;
            }

      return 0;

case WM_CHAR:
      /*----------------------------
      Обрабатываем нажатую клавишу.
      ----------------------------*/
      wParam = toupper ( wParam );
      switch (wParam)
            {
            case 194: /* Код русской буквы В */
                  SendMessage ( hWnd,
                                WM_COMMAND,
                                PLAY_PB,
                                0);
                  break;

            case 213: /* Код русской буквы Х */
                  SendMessage ( hWnd,
                                WM_COMMAND,
                                EXIT_PB,
                                0);
                  break;
            }

      return 0L;

case WM_COMMAND:
      /*--------------------
      Обработка команд меню.
      ----------------------*/

      switch (wParam)
            {
            case PLAY_PB:
                  /*-------------------------------------
                  Воспроизводим фрагмент звукового файла.
                  Диапазон воспроизведения:
                  От: Начало звукового файла.
                  До: Конец файла.
                  -------------------------------------*/

                  iPlayResult =
                  ts_PlayWave ( hWnd,
                                TS_START_OF_FILE,
                                TS_END_OF_FILE,
                                TS_IN_MILLISECONDS );
```

```
                    if ( iPlayResult != 1 )
                        {
                        MessageBox ( NULL,
                                     "Невозможно воспроизвести.",
                                     "Сообщение",
                                     MB_ICONINFORMATION );
                        }

                    return 0L;

               case EXIT_PB:
               case IDM_QUIT:
                    /*-------------------------------------
                    Пользователь выбрал команду Завершить.
                    -------------------------------------*/
                    DestroyWindow (hWnd);
                    return 0L;

               case IDM_ABOUT :
                    /*-------------------------------------
                    Пользователь выбрал команду О Программе.
                    -------------------------------------*/
                    DialogBox ( ghInst,
                                "AboutBox",
                                hWnd,
                                lpfnAboutDlgProc );
                    return 0;
               }/* конец switch (wParam) */
          case MM_MCINOTIFY:
               switch ( wParam )
                    {
                    case MCI_NOTIFY_SUCCESSFUL:
                         break;
                    case MCI_NOTIFY_ABORTED:
                         break;
                    case MCI_NOTIFY_SUPERSEDED:
                         break;
                    case MCI_NOTIFY_FAILURE:
                         MessageBox ( NULL,
                                      "Ошибка!",
                                      "Сообщение",
                                      MB_ICONINFORMATION );
                         break;
                    }/* конец switch() */

               return 0L;
          case WM_DESTROY:
               /*----------------------
               Закрываем сеанс "Wave".
               ----------------------*/
               ts_CloseWaveSession();
               PostQuitMessage (0);
               return 0;
          }/* конец switch (message) */
```

```
/*------------------------
 Сообщение не обработано.
 ------------------------*/
return DefWindowProc ( hWnd, message, wParam, lParam ) ;

}/* конец WndProc() */
/*================== конец WndProc() ====================*/

/*=========================
 ФУНКЦИЯ: AboutDlgProc()
 =========================*/
/*----------------------------------------
 ОПИСАНИЕ:
 Это процедура диалогового окна О Программе.
 ----------------------------------------*/
BOOL FAR PASCAL _export AboutDlgProc ( HWND hDlg,
                                       UINT message,
                                       UINT wParam,
                                       LONG lParam )
{
switch ( message )
       {
       case WM_INITDIALOG :
            return TRUE;

       case WM_COMMAND :
            switch ( wParam )
                   {
                   case IDOK :
                   case IDCANCEL :
                        EndDialog ( hDlg, 0 );
                        return TRUE;
                   }
       }
return FALSE ;
}/* Конец функции. */
/*================ Конец AboutDlgProc() ==============*/
```

Листинг 11.3. Hello2.rc

```
/*=========================================
 ИМЯ ФАЙЛА: Hello2.rc
 ----------

 НАЗНАЧЕНИЕ ФАЙЛА:
 -----------------
 Файл ресурсов.

 (C) Copyright Gurewich 1992, 1993

 =========================================*/

/*--------
 #include
 --------*/
#include <windows.h>
```

```
#include "Hello2.h"

/*---
 Меню
 ----*/
Hello2  MENU
BEGIN
   POPUP  "&Меню"
        BEGIN
           MENUITEM "&Завершить",    IDM_QUIT
           MENUITEM "&О Программе", IDM_ABOUT
        END
END

/*----------------------------------------
 Определение пиктограммы внешнего динамика.
 Имя файла: ExtSpkr.ico
 Имя пиктограммы: IconOfExtSpkr
 ----------------------------------------*/
IconOfExtSpkr ICON ExtSpkr.ico

/*---------------------
 Растровые изображения.
 ---------------------*/
BackGround    BITMAP Back2.bmp
/*--------------------------
 Диалоговое окно О Программе.
 --------------------------*/
AboutBox DIALOG 81, 43, 160, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Об этой программе"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON       "OK", IDOK, 64, 75, 40, 14
    CTEXT            "(C) Copyright Gurewich 1992, 1993", -1,
                      13, 47, 137, 18
    ICON             "IconOfExtSpkr", -1, 14, 12, 18, 20
END
/*-----------------------------------------
 Главное окно.
 Это окно появляется после запуска приложения.
 -----------------------------------------*/
Hello2Box DIALOG PRELOAD 59, 59, 221, 124

STYLE WS_MINIMIZEBOX | WS_POPUP    | WS_VISIBLE | WS_CAPTION |
                       WS_SYSMENU | WS_THICKFRAME
CAPTION "Hello2"
FONT 8, "MS Sans Serif"
CLASS "Hello2Class"
BEGIN
    PUSHBUTTON       "&Воспроизведение", PLAY_PB, 62, 17, 112, 31
    PUSHBUTTON       "Вы&ход", EXIT_PB, 98, 94, 40, 14
END
```

Листинг 11.4. Hello2.def

```
;====================================
; Файл определения модуля для Hello2.c
;====================================

NAME           Hello2

DESCRIPTION  'Программа Hello2. (C) Copyright Gurewich 1992, 1993'

EXETYPE        WINDOWS

STUB           'WINSTUB.EXE'

CODE   PRELOAD MOVEABLE DISCARDABLE

DATA   PRELOAD MOVEABLE MULTIPLE

HEAPSIZE       1024
STACKSIZE      8192
```

Листинг 11.5. Hello2.mak

```
#============
# Hello2.mak
#============

Hello2.exe : Hello2.obj Hello2.h Hello2.def Hello2.res
    link /nod Hello2, Hello2, NUL, \
         slibcew oldnames libw commdlg \
         c:\spSDK\TegoWlib\TSWlib, \
         Hello2.def
    rc -t Hello2.res

Hello2.obj : Hello2.c Hello2.h
   cl -c -G2sw -Ow -W3 -Zp Hello2.c

Hello2.res : Hello2.rc Hello2.h ExtSpkr.ico
   rc -r Hello2.rc
```

#include-файлы программы Hello2

Программа Hello2 включает в себя следующий оператор #include:

```
#include "c:\spSDK\TegoWlib\ts4Win.h"
```

Этот оператор служит для подключения файла ts4Win.h, расположенного в директории c:\spSDK\TegoWlib\ (в предыдущих главах мы подключали файл c:\spSDK\TegoWlib\sp4Win.h).

Для программ, использующих функции sp_ и определения sp_, нужно подключать файл c:\spSDK\TegoWlib\sp4Win.h, а для программ, использующих функции ts_ и определения ts_, — файл c:\spSDK\TegoWlib\ts4Win.h.

Имена всех функций из библиотеки TSWlib.lib начинаются с символов ts_: ts_- функции служат для управления звуковой платой, операциями MIDI, CD-устройствами и другими функциями мультимедиа и анимации. Библиотека TSWlib.lib, поставляемая на дискете в комплекте с книгой, представляет собой сокращенную версию и поэтому содержит ограниченное количество функций ts_. Однако этих функций достаточно, чтобы компилировать и компоновать все представленные в книге примеры программ.

Примечание:
ДЛЯ ПОЛЬЗОВАТЕЛЕЙ КОМПИЛЯТОРА ФИРМЫ MICROSOFT:

Файл c:\spSDK\TegoWlib\TegoWin.lib — это библиотека, содержащая sp_-функции, которые позволяют программе воспроизводить звук через встроенный динамик ПК.

Для того чтобы использовать функции из библиотеки TegoWin.lib, ваша программа должна содержать следующий оператор:

```
#include "c:\spSDK\TegoWlib\sp4Win.h"
```

Файл c:\spSDK\TegoWlib\TSWlib.lib — это библиотека, содержащая функции ts_, которые позволяют программам воспроизводить звук и выполнять различные задачи мультимедиа с помощью аппаратного обеспечения мультимедиа — звуковых плат, MIDI, CD, джойстиков и др.

Для использования ts_-функций из библиотеки TSWlib.lib ваша программа должна включать в себя следующий оператор:

```
#include "c:\spSDK\TegoWlib\ts4Win.h"
```

Если в вашей программе используются и sp_- и ts_-функции, то она должна включать в себя и файл sp4Win.h, и файл ts4Win.h.

Дискета, прилагаемая к данной книге, содержит сокращенные версии библиотек TegoWin.lib и TSWlib.lib.

Примечание:
ДЛЯ ПОЛЬЗОВАТЕЛЕЙ КОМПИЛЯТОРА ФИРМЫ BORLAND:

Файл c:\spSDK\TegoWlib\TegoWBL.lib — это библиотека, содержащая sp_-функции, которые позволяют воспроизводить звук через встроенный динамик ПК.

Для того чтобы использовать функции из библиотеки TegoWBL.lib, ваша программа должна содержать следующий оператор:

```
#include "c:\spSDK\TegoWlib\sp4Win.h"
```

Если ваша программа использует и sp_- и ts_-функции, то она должна включать в себя и файл sp4Win.h, и файл ts4Win.h.

Дискета, прилагаемая к данной книге, содержит сокращенные версии библиотек TegoWBL.lib и TS4BL.lib.

Make-файл программы Hello2

Поскольку программа Hello2 использует функции из библиотеки TSWlib.lib, то мы должны включить эту библиотеку в список библиотек, которые будут подсоединяться. Вы можете убедиться, что оператор компоновки в файле Hello2.mak действительно содержит ссылку на эту библиотеку:

```
link /nod Hello2, Hello2, NUL, \
    slibcew oldnames libw commdlg \
    c:\spSDK\TegoWlib\TSWlib, \
```

Файл Hello2.bmk — это make-файл программы Hello2, который используется компилятором фирмы Borland.

То же относится и к библиотеке TS4BL.lib: программа Hello2 использует функции из этой библиотеки, следовательно, ее также нужно включить в упомянутый выше список библиотек. Просмотрев файл Hello2.bmk, вы можете убедиться, что оператор компоновки в этом файле содержит соответствующую ссылку:

```
tlink /c /n Tw /L\borlandc\lib\ c0ws Hello2, Hello2, NUL, \import
maths cws c:\spSDK\TegoWlib\TS4BL.lib, \
```

Функция WinMain программы Hello2

Функция WinMain() программы Hello2 содержит цикл обработки сообщений. Это цикл типа Generic1:

```
while ( GetMessage ( &msg, NULL, 0, 0 ) )
      {
      TranslateMessage ( &msg );
      DispatchMessage  ( &msg );
      }
```

Преимущество применения звуковой платы (кроме улучшения качества звука) заключается в том, что программа должна лишь послать ей команду воспроизвести звуковой фрагмент, а звуковая плата всю работу по воспроизведению сделает сама, без какой-либо помощи со стороны ПК. Таким образом, при воспроизведении с помощью звуковой платы процессор свободен и может выполнять другие задачи.

Применение диалогового окна в качестве главного окна приложения

Программа Hello2 в качестве главного окна приложения использует диалоговое окно. Для этого следует установить значение элемента wc.cbWndExtra на DLGWINDOWEXTRA:

```
wc.cbWndExtra    = DLGWINDOWEXTRA ; /* Диалоговое окно как */
                                    /* главное окно.       */
```

а элемента wc.lpszClassName — на Hello2Class:

```
wc.lpszClassName = "Hello2Class"; /* Как указано в разделе CLASS */
                                  /* файла .RC.                  */
```

В файле Hello2.rc класс диалогового окна Hello2Box определяется следующим образом:

```
Hello2Box DIALOG PRELOAD 59, 59, 221, 124
.....................................
.....................................
CLASS "Hello2Class"
```

```
BEGIN
.....................................
.....................................
END
```

Это окно является главным окном программы.

Для того чтобы создать в главном диалоговом окне желтый фон, загружаем растровое изображение фона BackGround и, используя это растровое изображение, создаем кисть:

```
hBitmapOfBkGnd = LoadBitmap ( ghInst, "BackGround" );
hBrushOfBkGnd  = CreatePatternBrush ( hBitmapOfBkGnd );
```

Элемент wc.hbrBackground определяется следующим образом:

```
wc.hbrBackground = hBrushOfBkGnd;   /* Кисть, которую мы создали на */
                                    /* основе растрового изображения.*/
```

Пиктограмма программы Hello2

Пиктограмма программы Hello2 определяется следующим образом:

```
wc.hIcon = LoadIcon ( ghInst, "IconOfExtSpkr" ) ;
```

Пиктограмма IconOfExtSpkr определена в файле ресурсов Hello2.rc как ExtSpkr.ico:

```
IconOfExtSpkr ICON ExtSpkr.ico
```

Эта пиктограмма содержит изображение внешнего динамика, поскольку для работы программы требуется такой динамик. Такую пиктограмму можно создать с помощью редактора Image Editor, поставляемого вместе с SDK (Software Develompent Kit) для Windows фирмы Microsoft, либо с помощью программы Resource Workshop фирмы Borland (см. рис. 11.10).

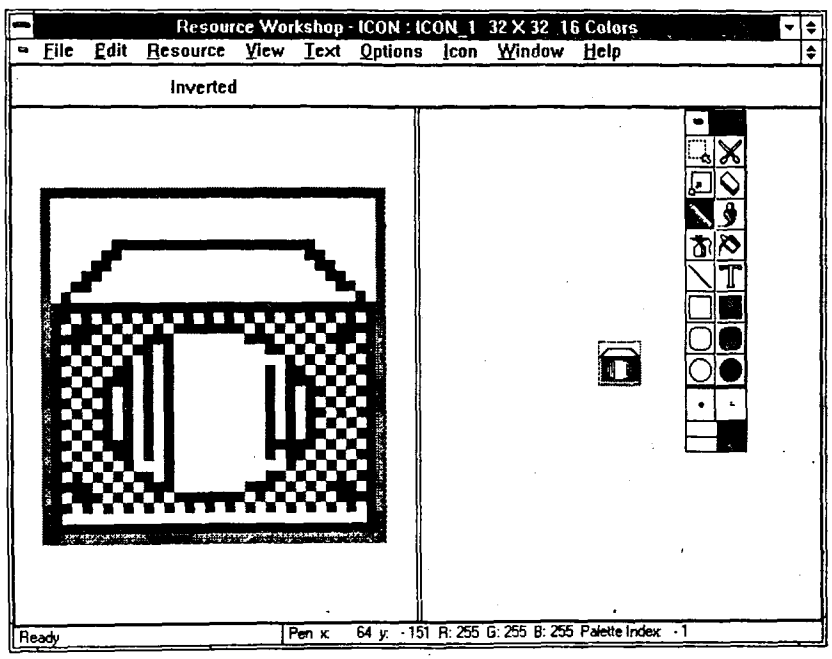


Рис. 11.10. Пиктограмма ExtSpkr.ico

Функция WndProc() программы Hello2

Функция WndProc() программы Hello2 обрабатывает различные события, которые соответствуют событиям, происходящим в процессе выполнения программы.

Вариант WM_CREATE программы Hello2

В варианте WM_CREATE мы получаем lpfnAboutDlgProc диалогового окна **О Программе**, а затем определяем, установлена ли в системе Windows-совместимая звуковая плата.

Определение наличия в системе звуковой платы

Чтобы определить, содержит ли система Windows-совместимую звуковую плату, которая может воспроизводить файлы .WAV, используют ts_-функцию ts_WaveDeviceCanPlay(). Если функция ts_WaveDeviceCanPlay() возвращает 0, то появляется диалоговое окно, сообщающее пользователю о том, что его ПК не оснащен Windows-совместимой звуковой платой:

```
/*-----------------------------------------
Существует ли в системе звуковая плата,
необходимая для воспроизведения файлов .WAV ?
-----------------------------------------*/
if ( ts_WaveDeviceCanPlay() == 0 )
   {
   MessageBox ( NULL,
                "Звуковая плата не обнаружена!",
                 "Сообщение",
                 MB_ICONSTOP );

   /*-------------------
   Завершаем приложение.
   -------------------*/
   SendMessage ( hWnd,
                 WM_DESTROY,
                 0,
                 0 );
   return 0;
}
```

Примечание:

Имя функции: ts_WaveDeviceCanPlay()

Описание: Эта функция определяет, оснащен ли ПК Windows-совместимой звуковой платой, которая может воспроизводить файлы .WAV.

Прототип: int ts_WaveDeviceCanPlay (void);

Параметры: Отсутствуют.

Возвращаемое значение: Целое. Значение 1 указывает на то, что система оснащена Windows-совместимой звуковой платой, которая может воспроизводить файлы .WAV; значение 0 — что такая плата в системе не установлена.

Функция ts_WaveDeviceCanPlay() возвращает 0, если установленная плата не является Windows-совместимой или один из необходимых драйверов отсутствует, или звуковой платы просто нет.

ts_-функции

В этой и последующих главах вы будете встречать и применять функции из библиотеки TSWlib.lib или из библиотеки TSWBL.lib, связанные с мультимедиа. Эти функции начинаются с символов ts_ и включают в себя слово *Wave*, (например ts_WaveDeviceCanPlay(), ts_OpenWaveSession()) — это означает, что функции обрабатывают звуковые файлы .WAV. На дискете, которая прилагается к данной книге, поставляется сокращенная версия библиотеки TSWlib.lib (или TSWBL.lib). Полная версия библиотеки TSWlib.lib содержит больше функций ts_Wave; она включает в себя и функции для управления другими устройствами мультимедиа. Так, функции ts_CD служат для управления CD-устройствами, функции ts_MIDI — MIDI-устройствами и т.д.

Выбор устройства для воспроизведения звука

Созданная нами программа Hello2 в случае, если компьютер не оснащен Windows-совместимой звуковой платой, попросту завершает свою работу. Однако почему бы нам не расширить возможности этой программы? Мы можем определить в ней специальный флаг — переменную (статическую или глобальную), значение которой будет указывать, какое устройство должно использоваться для воспроизведения звука: звуковая плата или встроенный динамик. Этот флаг будет устанавливаться в зависимости от значения, возвращаемого функцией ts_WaveDeviceCanPlay(). В соответствии со значением флага будет выполняться либо фрагмент программы для воспроизведения звука через динамик, либо другой фрагмент — для воспроизведения звука с помощью звуковой платы. Таким образом мы получим универсальный .EXE-файл, который смогут использовать как те, кто имеет звуковую плату, так и те, кто ее не имеет. Еще одно достоинство этой программы: она не задает пользователю лишних вопросов о том, инсталлирована ли на его компьютере звуковая плата. И наконец, описание программы может включать, например, следующее:

```
                ЗВУК ПОДДЕРЖИВАЕТСЯ!

Эта программа воспроизводит звук либо через звуковую
плату (если она установлена на вашем компьютере), либо
через встроенный динамик БЕЗ какого-либо дополнительного
аппаратного или программного обеспечения.
```

Впрочем, такое заявление в программе делать совсем не обязательно. В этом случае пользователи вашей программы будут приятно удивлены, обнаружив, что она рассчитана на работу с различными устройствами для воспроизведения звука.

Если вы не сообщите своим пользователям о том, что ваша программа или демонстрационный пример "разговаривает" настоящим человеческим голосом, то они будут поражены и очарованы, *услышав* вашу программу после запуска. Пользователи, не имеющие звуковой платы, будут еще более удивлены, услышав человеческий голос, воспроизводимый через обычный встроенный динамик ПК, — ведь большинство пользователей ПК и не подозревают о том факте, что динамик компьютера может воспроизводить что-либо кроме примитивных сигналов.

Открытие сеанса WAVE

После того как программа определила, что в системе установлена Windows-совместимая звуковая плата, открывается (для этого используется функция ts_OpenWaveSession()) звуковой файл .WAV:

```
/*-----------------------
Открываем звуковой сеанс.
-----------------------*/
lstrcpy( sWAVFileName, "c:\\spSDK\\WAV\\Hello2.wav");
iOpenResult =
ts_OpenWaveSession ( sWAVFileName );

if ( iOpenResult != 1 )
   {
   lstrcpy ( sMessage, "Невозможно открыть сеанс для : ");
   lstrcat ( sMessage, sWAVFileName );
   MessageBox ( NULL,
                sMessage,
                "Сообщение",
                MB_ICONINFORMATION );

   /*-------------------
   Завершаем приложение.
   -------------------*/
   SendMessage ( hWnd,
                 WM_DESTROY,
                 0,
                 0 );
   return 0;
}
```

Примечание:

Имя функции: ts_OpenWaveSession ()

Описание: Эта функция открывает Wave-сеанс.

Прототип: int ts_OpenWaveSession (LPSTR lpstrFileName);

Параметры: Строка, завершаемая значением NULL, которая содержит имя файла .WAV, подлежащего открытию.

Возвращаемое значение: Целое. Значение 1 указывает на то, что Wave-сеанс открыт успешно. Любые другие значения указывают на ошибку.

Функция ts_OpenWaveSession() может не открыть Wave-файл, если указанного файла не существует, если Wave-файл поврежден, отсутствуют необходимые драйверы, а также из-за других ошибок.

Автономные программы

Подобно программам, описанным в предыдущих главах, приложения Windows, которые поддерживают работу со звуковой платой, могут быть как автономными, так и неавтономными звуковыми программами. Для преобразования неавтономных звуковых программ в автономные воспользуйтесь утилитой, подключающей к вашей программе звуковые файлы .WAV. Эта утилита входит в состав редактора Sound Editor фирмы TS. Поскольку мы предполагаем, что у вас нет копии такого редактора, то все программы, речь о которых идет в этой и последующих главах, построены как неавтономные.

Функция ts_PlayWave()

Для воспроизведения Wave-файла применяется функция ts_PlayWave(). В числе параметров этой функции указываются начальная координата, конечная координата и единица измерения координат. Например, для воспроизведения фрагмента звукового Wave-файла от координаты 3 мс до координаты 432 мс необходимо использовать следующий оператор:

```
ts_PlayWave ( hWnd, 3, 432, TS_IN_MILLISECONDS );
```

Четвертый параметр — TS_IN_MILLISECONDS — указывает на то, что координаты (второй и третий параметры) заданы в миллисекундах.

Примечание:

Имя функции: ts_PlayWave()

Описание: Эта функция воспроизводит фрагмент Wave-файла через Windows-совместимую звуковую плату

Прототип: int ts_PlayWave (HWND hWnd,
long lStartingCoordinates,
long lEndingCoordinates,
int iUnits);

Возвращаемое значение: Целое. Равно 1, если фрагмент звукового файла может быть воспроизведен. Значение, отличное от 1, указывает на то, что фрагмент звукового файла воспроизвести невозможно.

Параметры:

HWND hWnd:	В процессе воспроизведения все сообщения, связанные с этим процессом, отображаются в окне hWnd.
long lStartingCoordinates:	Начальная координата фрагмента, который будет воспроизводиться.
long lEndingCoordinates:	Конечная координата фрагмента, который будет воспроизводиться.
int iUnits:	Единица измерения координат. Если второй и третий параметры заданы в миллисекундах, то параметр iUnits должен быть равен TS_IN_MILLISECONDS.

Сообщение MM_MCINOTIFY

Как уже отмечалось, после того как наша программа вызвала функцию ts_PlayWave(), звуковая плата делает все сама, воспроизводя Wave-файл без какой-либо дополнительной помощи.

Чтобы программа узнала о том, что воспроизведение Wave-файла завершено, необходимо передать функции ts_PlayWave() hWnd нашей программы. В том случае, когда воспроизведение фрагмента звукового файла закончится или по какой-либо причине будет прервано, программа получит сообщение MM_MCINOTIFY.

Ниже показано, как сообщение MM_MCINOTIFY обрабатывается в функции WndProc():

```
case MM_MCINOTIFY:
switch ( wParam )
       {
       case MCI_NOTIFY_SUCCESSFUL:
            break;
```

Вариант WM_CHAR программы Hello2

В варианте WM_CHAR код символа клавиши, нажатой пользователем, преобразуется в соответствующий код символа верхнего регистра и проверяется. Если была нажата клавиша [B], то будет использована функция SendMessage() для посылки сообщения WM_COMMAND с wParam, равным PLAY_PB, что дает эффект нажатия командной кнопки **Воспроизведение**.

Подобным образом, если обнаружится, что нажата клавиша [x], будет вызвана функция SendMessage() для посылки сообщения с wParam, равным EXIT_PB, что приведет к эффекту нажатия командной кнопки **Выход**:

```
case WM_CHAR:
     /*---------------------------
     Обрабатываем нажатую клавишу.
     ---------------------------*/
     wParam = toupper ( wParam );
     switch (wParam)
          {
          case 194: /* Код русской буквы B */
               SendMessage ( hWnd,
                             WM_COMMAND,
                             PLAY_PB,
                             0);
               break;

          case 213: /* Код русской буквы X */
               SendMessage ( hWnd,
                             WM_COMMAND,
                             EXIT_PB,
                             0);
        break;
   }
return 0L;
```

Вариант WM_COMMAND программы Hello2

В варианте WM_COMMAND обрабатывается сообщение PLAY_PB, которое было сгенерировано в ответ на нажатие пользователем командной кнопки **Воспроизведение**:

```
case PLAY_PB:
     /*-------------------------------------
     Воспроизводим фрагмент звукового файла.
     Диапазон воспроизведения:
     От: Начало звукового файла.
     До: Конец файла.
     -------------------------------------*/
     iPlayResult =
     ts_PlayWave ( hWnd,
                   TS_START_OF_FILE,
                   TS_END_OF_FILE,
                   TS_IN_MILLISECONDS );

     if ( iPlayResult != 1 )
        {
        MessageBox ( NULL,
                     "Невозможно воспроизвести фрагмент
                      звукового файла.",
                     "Сообщение",
                     MB_ICONINFORMATION );
        }
  return 0L;
```

```
case MCI_NOTIFY_ABORTED:
     break;
case MCI_NOTIFY_SUPERSEDED:
     break;
case MCI_NOTIFY_FAILURE:
     MessageBox ( NULL,
                  "Ошибка!",
                  "Сообщение",
                   MB_ICONINFORMATION );
     break;
}/* конец switch() */

     return 0L;
```

В этом фрагменте программы обрабатываются все возможные значения wParam, соответствующие сообщению MM_MCINOTIFY. Очевидно, что в случае успешного завершения значение wParam равно MCI_NOTIFY_SUCCESSFUL. Другие значения wParam указывают на то, что воспроизведение было прервано, произошло "вытеснение" при воспроизведении или ошибка.

В программе Hello сообщения MM_MCINOTIFY никак не обрабатываются, т.е. в любом случае происходит выход из оператора. Эти сообщения возникают, когда воспроизведение прерывается функцией ts_StopWave или начинается новый процесс воспроизведения с помощью функции ts_PlayWave() до завершения текущего воспроизведения (функция ts_StopWave() рассматривается ниже).

Если принято сообщение MCI_NOTIFY_SUPERSEDED, то это означает, что при воспроизведении произошло "вытеснение", т.е. воспроизведение прервано посылкой звуковой плате другой команды. Существует несколько причин "вытеснения" и, следовательно, мы можем включить в программу обработку таких ситуаций с целью предотвращения посылки звуковой плате команд, приводящих к "вытеснению". Окно следующей программы содержит командные кнопки **Играть**, **Останов** и **Запись**. Для предотвращения сообщения MCI_NOTIFY_SUPERSEDED мы запрещаем командную кнопку **Запись** до тех пор, пока идет процесс воспроизведения, а командную кнопку **Играть** — до тех пор, пока идет процесс записи.

Отдельный сеанс для каждой звуковой платы

Большинство звуковых плат не могут одновременно записывать и воспроизводить — для этого в компьютере должны быть установлены две звуковые платы (например, в мультимедиа-программе телеконференции необходимо одновременно посылать и принимать аудиосигналы).

Вариант WM_DESTROY программы Hello2

Перед завершением программы мы закрываем звуковой файл Wave применением функции ts_CloseWaveSession():

```
case WM_DESTROY:
     /*---------------------
     Закрываем сеанс "Wave".
     ----------------------*/
     ts_CloseWaveSession();
     PostQuitMessage (0);
     return 0;
```

Примечание:

Имя функции: ts_CloseWaveSession();

Описание: Эта функция закрывает звуковой сеанс Wave;

Прототип: int ts_CloseWaveSession (void);

Возвращаемое значение: Всегда возвращает значение 1.

Параметры: Отсутствуют.

Программа Record

Программа Record для записи и воспроизведения звуковых Wave-файлов использует ts_-функции.

Мы рекомендуем откомпилировать, скомпоновать и выполнить программу Record перед ее изучением — это позволит лучше понять возможности программы.

Компиляция, компоновка и выполнение программы Record

Для компиляции и компоновки программы компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
NMAKE Record.mak    [Enter]
```

Для компиляции и компоновки программы компилятором фирмы Borland необходимо:

- Войти в директорию c:\spSDK\Samp4Win.
- После приглашения DOS ввести:

```
MAKE -f Record.bmk    [Enter]
```

Перед выполнением программы Record проверьте, установлены ли на вашем компьютере Windows-совместимая звуковая плата и все необходимые драйверы.

Для выполнения программы Record:

- Выберите из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- В диалоговом окне **Пролистать** выберите файл

```
c:\spSDK\Samp4Win\Record.exe
```

Появится главное окно программы, изображенное на рис. 11.11.

Меню программы Record содержит две команды — **Завершить** и **О Программе** (см. рис. 11.12). Если пользователь выбирает команду **Завершить**, то программа завершается, если команду **О Программе** — то, как показано на рис. 11.13, появляется диалоговое окно команды **О Программе**.

Главное окно программы Record содержит четыре командные кнопки — **Играть**, **Останов**, **Запись** и **Выход**. Для записи файла с расширением .WAV нажмите командную кнопку **Запись** и произнесите в микрофон слова, которые вы хотите записать. Для остановки записи нажмите командную кнопку **Останов**, а для воспроизведения файла, который вы записали, — командную кнопку **Играть**. Для закрытия главного окна программы нажмите командную кнопку **Выход**.

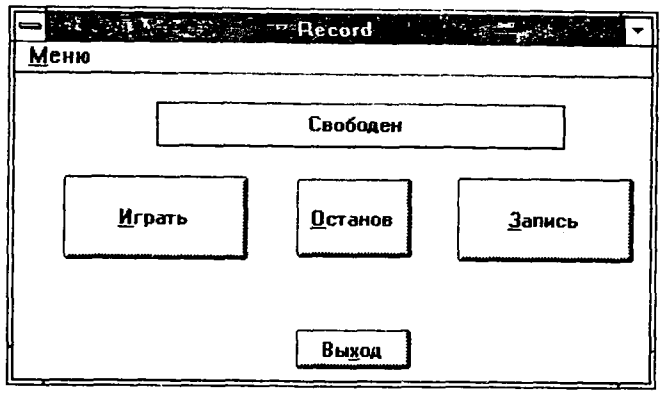


Рис. 11.11. Главное окно программы Record

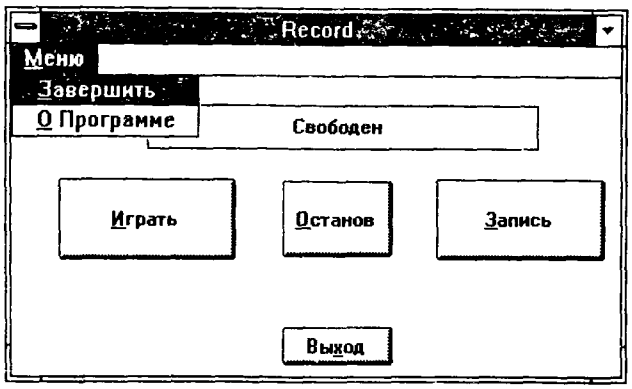


Рис. 11.12. Меню программы Record

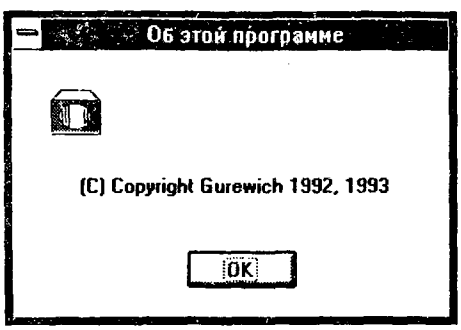


Рис. 11.13. Диалоговое окно команды О Программе программы Record

В процессе воспроизведения разрешено применять мышь, и пользователь может переключиться в другое приложение Windows.

Программа Record не позволяет сохранить записанный Wave-файл. Возможность сохранения записи включена в следующую программу, приведенную в этой главе.

Файлы программы Record

Программа Record состоит из следующих файлов:

Record.h	#include-файл программы
Record.c	Текст программы на C
Record.rc	Файл ресурсов программы
Record.def	Файл определения модуля программы
Record.mak	Make-файл программы
ExtSpkr.ico	Пиктограмма программы

Эти файлы доступны для вас в директории c:\spSDK\Samp4Win\.

Вариант WM_CREATE программы Record

В варианте WM_CREATE мы определяем характеристики звуковой платы. Функция ts_WaveDeviceCanPlay() определяет, способна ли звуковая плата, установленная в вашем персональном компьютере, воспроизводить Wave-файлы (.WAV).

Функция ts_WaveDeviceCanRecord() определяет, способна ли звуковая плата, установленная в вашем ПК, записывать Wave-файлы.

Примечание:

Имя функции: tsWaveDeviceCanRecord()

Описание: Эта функция определяет, оборудован ли ваш ПК Windows-совместимой звуковой платой, которая может записывать файлы с расширением .WAV.

Прототип: int tsWaveDeviceCanRecord (void);

Параметры: Отсутствуют.

Возвращаемое значение: Целое. Значение 1 указывает на то, что система оснащена Windows-совместимой звуковой платой, способной записывать файлы с расширением .WAV; значение 0 — на то, что в системе такая плата не установлена.

Если в системе установлена звуковая плата, способная как записывать, так и воспроизводить файлы .WAV то звуковой сеанс открывается функцией ts_OpenWaveSession(). Поскольку звуковой сеанс открывается для файла, которого еще не существует, то в качестве имени Wave-файла указываем нулевую строку:

```
/*--------------------------
Открываем сеанс мультимедиа.
----------------------------*/
lstrcpy( sWAVFileName, "");
iOpenResult =
ts_OpenWaveSession ( sWAVFileName );

if ( iOpenResult != 1 )
   {
   lstrcpy ( sMessage,
             "Невозможно открыть сеанс.");
   MessageBox ( NULL,
                sMessage,
                "Сообщение",
                MB_ICONINFORMATION );
```

```
/*-------------------
Завершаем приложение.
---------------------*/
SendMessage ( hWnd,
              WM_DESTROY,
              0,
              0 );
return 0;
```

Обработка запросов на воспроизведение в варианте WM_COMMAND

Когда пользователь нажимает командную кнопку **Играть**, выполняется вариант PLAY_PB варианта WM_COMMAND:

```
case PLAY_PB:
     /*-----------------------------------------
     Пользователь нажал командную кнопку Играть.
     -----------------------------------------*/
     /*---------------------------
     Запрет кнопок Играть и Запись.
     ---------------------------*/
     DisablePlayAndRecord ( hWnd );

     /*-----------------------------
     Отображение заголовка состояния.
     (Процесс воспроизведения)
     -----------------------------*/
     DisplayStatus( hWnd, 'P' );

     /*---------------------------
     Воспроизводим звуковой раздел.
     Диапазон воспроизведения:
     От: Начало звукового файла.
     До: Конец файла.
     ---------------------------*/
     iPlayResult =
     ts_PlayWave ( hWnd,
                   TS_START_OF_FILE,
                   TS_END_OF_FILE,
                   TS_IN_MILLISECONDS );

     if ( iPlayResult != 1 )
        {
        /*---------------------------
        Звуковая плата не может принять
        команду воспроизведения.
        ---------------------------*/
        /*-----------------------------
        Разрешение кнопок Играть и Запись.
        -----------------------------*/
        EnablePlayAndRecord ( hWnd );

        /*---------------------------
        Отображение заголовка состояния.
        (Свободен)
        ---------------------------*/
        DisplayStatus ( hWnd, 'I' ) ;
        }
     return 0L;
```

Как только функция ts_PlayWave() начнет выполняться, звуковая плата будет воспроизводить файл, и компьютер на все время, необходимое для воспроизведения, свободен для выполнения других задач.

Если пользователь во время воспроизведения не нажал командную кнопку **Останов**, то воспроизведение автоматически завершается по достижении конца диапазона воспроизведения, указанного в функции ts_PlayWave(). Наша программа будет извещена об этом событии сообщением MM_MCINOTIFY.

Обработка запросов на запись в варианте WM_COMMAND

Подобным же образом обрабатываем сообщения от командной кнопки **Запись**. В варианте RECORD_PB разрешается использовать в процессе записи только командные кнопки **Останов** и **Выход**.

Перед началом записи следует удалить текущий Wave-файл (если он существует). Для этого используется функция ts_DeleteWave(). После этого выполняется функция ts_RecordWave(). Ниже приведен вариант RECORD_PB:

```
case RECORD_PB:
     /*----------------------------------------
     Пользователь нажал командную кнопку Запись.
     ----------------------------------------*/

     /*----------------------------
     Запрет кнопок Играть и Запись.
     ----------------------------*/
     DisablePlayAndRecord ( hWnd );

     /*----------------------------
     Отображение заголовка состояния.
     (Процесс записи)
     ----------------------------*/
     DisplayStatus( hWnd, 'R' );

     /*-------------------------------------------
     Удаление предыдущей записи (если она существует).
     -------------------------------------------*/
     ts_DeleteWave ( TS_START_OF_FILE,
                     TS_END_OF_FILE,
                     TS_IN_MILLISECONDS );

     /*--------------------------------
     Запись фрагмента звукового файла.
     Диапазон записи:
     От: Начало звукового файла.
     До: 30 секунд.
     --------------------------------*/
     iRecordResult =
     ts_RecordWave ( hWnd,
                     TS_START_OF_FILE,
                     30000L,
                     TS_IN_MILLISECONDS );

     if ( iRecordResult != 1 )
        {
        /*-----------------------------------------
        Звуковая плата не может принять команду Запись.
        -----------------------------------------*/
```

```
        /*--------------------------------
        Разрешение кнопок Играть и Запись.
        --------------------------------*/
        EnablePlayAndRecord ( hWnd );
        }
return 0L;
```

Функция ts_RecordWave()

Функция ts_RecordWave() используется для того, чтобы послать на звуковую плату команду начать записывать файл Wave. Диапазон записи указан в параметрах вызова функции.

Примечание:

Имя функции: ts_RecordWave()

Описание: Эта функция записывает файл с расширением .WAV.

Прототип: int ts_RecordWave (HWND hWnd,
long lStartingPoint,
long lEndingPoint,
int iUnits);

HWND hWnd: Указатель окна, которое будет принимать сообщение MM_MCINOTIFY.
long lStartingPoint: Позиция, с которой начинается запись.
long lEndingPoint: Конечная позиция диапазона записи.
int iUnits: Единица измерения lStartingPoint и lEndingPoint.

Возвращаемое значение: Целое. Значение 1 указывает на то, что команда записи была принята успешно; значение, отличное от 1, — на то, что звуковая плата не может принять команду **Запись**.

Функция ts_RecordWave() работает так же, как функция ts_PlayWave(), т.е. после ее вызова звуковая плата производит запись, и ПК в это время свободен для выполнения других задач.

Если пользователь не нажал во время записи командную кнопку **Останов**, то запись автоматически завершится после достижения конца диапазона, заданного параметрами функции ts_RecordWave(). Программа будет извещена о том, что конец записываемого диапазона достигнут, сообщением MM_MCINOTIFY.

Функция ts_DeleteWave()

В варианте RECORD_PB перед выполнением функции ts_RecordWave() выполняется функция ts_DeleteWave(). Функция ts_DeleteWave() удаляет предыдущую звуковую запись, если она существует. Учтите, что если перед выполнением функции ts_RecordWave() вы не выполните функцию ts_DeleteWave(), то новая звуковая запись будет автоматически вставлена в существующую.

Примечание:

Имя функции: ts_DeleteWave()

Описание: Эта функция удаляет фрагмент Wave-файла.

Прототип: int ts_DeleteWave (long lStartingPoint,
long lEndingPoint,
int iUnits);

long lStartingPoint:	Начальная позиция, с которой будет удален фрагмент звукового файла.
long lEndingPoint:	Конечная позиция удаляемого фрагмента звукового файла.
int iUnits:	Единица измерения lStartingPoint и lEndingPoint.

Возвращаемое значение: Целое. Значение 1 указывает на то, что удаление было выполнено успешно; значение, отличное от 1, — на то, что удаление не было выполнено.

В варианте RECORD_PB мы удаляем весь файл .WAV:

```
/*-----------------------------------------------
Удаление предыдущей записи (если она существует).
-----------------------------------------------*/
ts_DeleteWave ( TS_START_OF_FILE,
                TS_END_OF_FILE,
                TS_IN_MILLISECONDS );
```

Ограничение длительности звучания записи

В варианте RECORD_PB в функции ts_RecordWave() длительность звучания записи ограничена значением 30 с (30 с = 30000 мс):

```
iRecordResult =
ts_RecordWave ( hWnd,
                TS_START_OF_FILE,
                30000L,
                TS_IN_MILLISECONDS );
```

Очевидно, что для такого ограничения есть веские основания. Предположим, что пользователь начал запись, а затем, не останавливая ее, вышел из комнаты. Возвратившись, он может обнаружить, что запись заполнила весь жесткий диск! Такое возможно потому, что запись сохраняется на жестком диске в виде временного файла. Следовательно, длительность звучания записи всегда целесообразно ограничивать.

После того как запись будет завершена, функция ts_RecordWave() должна совершить еще некоторые действия. Если в ОЗУ отсутствует место для записи, то функция ts_RecordWave() сохранит эту запись на жестком диске.

Вставка новой записи в существующий файл .WAV

В программе Record мы удаляем предыдущую запись, а затем запускаем новую с позиции от 0 до 30 секунд максимум (или до тех пор, пока пользователь не нажмет командную кнопку **Останов**). Таким образом, вставка записей в этой программе не производится, но немного поэкспериментировать со вставкой вы можете.

Предположим, что у нас есть Wave-файл с длительностью звучания 30 с (от 0 до 30 с). Если при вызове функции ts_RecordWave() вы указали, что диапазон записи — от 10 до 20 с, и предварительно не была выполнена функция ts_DeleteWave(), то конечным результатом будет файл .WAV с длительностью звучания 40 с. Новая запись (с 10-ой по 20-ю секунду) будет вставлена в исходную. Вы можете выполнить вставку фрагмента звукового файла в любую позицию существующей записи. При этом значения диапазона записи должны иметь смысл. Например, если текущая запись занимает место от 0 до 40 с, то нельзя указать диапазон вставки от 50 до 60 с.

Остановка записи и воспроизведения

Если пользователь нажимает командную кнопку **Останов**, то выполняется функция ts_StopWave(), которая останавливает как процесс воспроизведения, так и процесс записи:

```
case STOP_PB:
     /*--------------------------------
     Пользователь нажал кнопку Останов.
     --------------------------------*/
     /*--------------------------------
     Разрешение кнопок Играть и Запись.
     --------------------------------*/
     EnablePlayAndRecord ( hWnd );
     /*--------------------------------
     Отображение заголовка состояния.
     (Свободен)
     --------------------------------*/
     DisplayStatus( hWnd, 'I' );

     ts_StopWave();

return 0L;
```

Примечание:

Имя функции: ts_StopWave()

Описание: Эта функция останавливает запись или воспроизведение (которые выполняются в данный момент).

Прототип: int ts_StopWave (void);

Параметры: Никаких.

Возвращаемое значение: Целое. Значение 1 указывает на то, что звуковая плата приняла команду **Останов**; значение, отличное от 1, — на то, что звуковая плата не приняла команду.

Вариант MM_MCINOTIFY

Сообщение MM_MCINOTIFY генерируется в тех случаях, когда существует сообщение от звуковой платы. Например, если звуковая плата завершает запись или воспроизведение, то функция WndProc() программы Record получает сообщение MM_MCINOTIFY.

В программе Record предусмотрен фрагмент для исключения возможности посылки звуковой плате нежелательной команды: при воспроизведении или при записи пользователь не может нажать кнопку **Играть** либо кнопку **Запись** — это запрещено.

Используя при создании программы такой способ, мы избегаем необходимости писать операторы для обработки различных сообщений MCI_NOTIFY_. Фактически в программе Record сообщение MM_MCINOTIFY принимается только тогда, когда завершен процесс записи или воспроизведения. После того как это сообщение принято, мы разрешаем командные кнопки **Играть** и **Запись**:

```
case MM_MCINOTIFY:
     /*--------------------------------
     Разрешение кнопок Играть и Запись.
     --------------------------------*/
     EnablePlayAndRecord ( hWnd );
```

```
        /*-----------------------------
        Отображение заголовка состояния.
        (Свободен)
        -----------------------------*/
        DisplayStatus ( hWnd, 'I' );

        switch ( wParam )
               {
               case MCI_NOTIFY_SUCCESSFUL:
                    break;

               case MCI_NOTIFY_ABORTED:
                    break;

               case MCI_NOTIFY_SUPERSEDED:
                    break;

               case MCI_NOTIFY_FAILURE:
                    MessageBox ( NULL,
                                 "Ошибка!",
                                 "Сообщение",
                                 MB_ICONINFORMATION );
                    break;

               }/* конец switch() */
    return 0L;
```

Каждый вариант MCI_NOTIFY_ содержит только один оператор break. Эти варианты включаются в программу Record для того, чтобы проиллюстрировать существование указанных сообщений. Такие сообщения особенно полезны в процессе разработки программы, причем для упрощения отладки может быть использован оператор MessageBox().

Вариант WM_DESTROY программы Record

В варианте WM_DESTROY выполняется функция ts_CloseWaveSession(), закрывающая звуковой сеанс:

```
case WM_DESTROY:
     /*---------------------
     Закрываем сеанс "Wave".
     ----------------------*/
     ts_CloseWaveSession();
     PostQuitMessage (0);
     return 0;
```

Частота дискретизации записи и дополнительные ts_-функции

Как вы могли заметить, в программе Record не указана частота дискретизации записи. Следовательно, запись выполнялась с частотой дискретизации, заданной по умолчанию (11025 Гц).

Полная версия библиотеки TS включает в себя дополнительные ts_-функции, которые дают возможность выполнять запись с любой желаемой частотой дискретизации; она содержит также и другие мощные ts_-функции, позволяющие управлять почти всеми возможными аспектами записи и воспроизведения.

Программа Record2

Сейчас мы расширим программу Record — с тем, чтобы дать возможность пользователю сохранять запись в файле .WAV, а также загружать и воспроизводить файлы .WAV. Новая расширенная программа называется Record2.

Компиляция, компоновка и выполнение программы Record2

Для компиляции и компоновки программы компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
NMAKE Record2.mak    [Enter]
```

Для компиляции и компоновки программы компилятором фирмы Borland необходимо:

- Войти в директорию c:\spSDK\Samp4Win.
- После приглашения DOS ввести:

```
MAKE -f Record2.bmk    [Enter]
```

Перед выполнением программы Record2 убедитесь в том, что на вашем ПК установлены Windows-совместимая звуковая плата и все необходимые драйверы.

Для выполнения программы Record2:

- Выберите из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- В диалоговом окне **Пролистать** выберите файл

```
c:\spSDK\Samp4Win\Record2.exe
```

Появится главное окно программы Record2, изображенное на рис. 11.14. Строка меню программы содержит два меню — **Файл** и **Выход** (рис. 11.15 и 11.16).

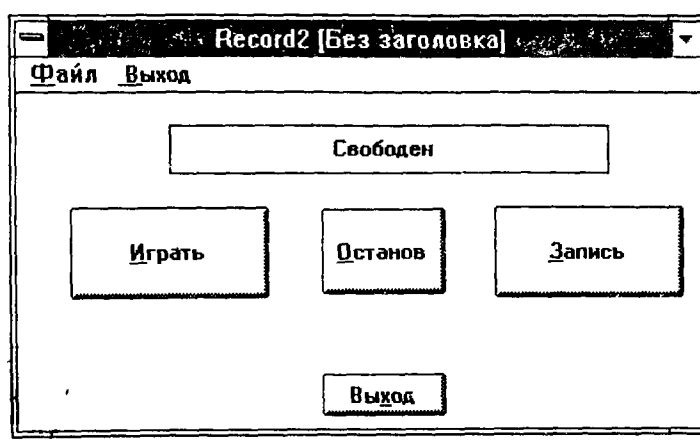


Рис. 11.14. Главное окно программы Record2

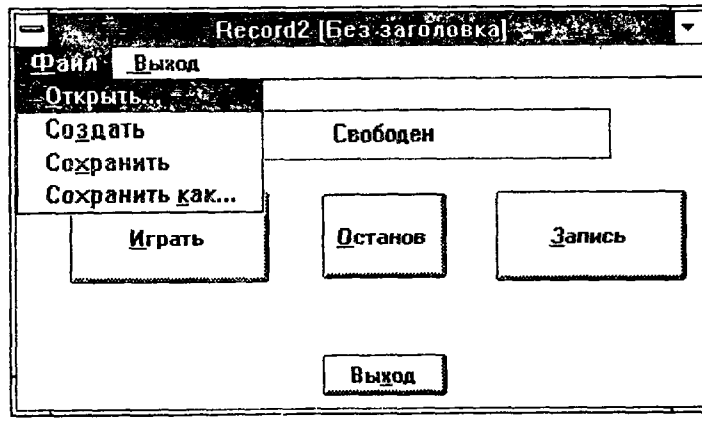


Рис. 11.15. Меню Файл главного окна программы Record2

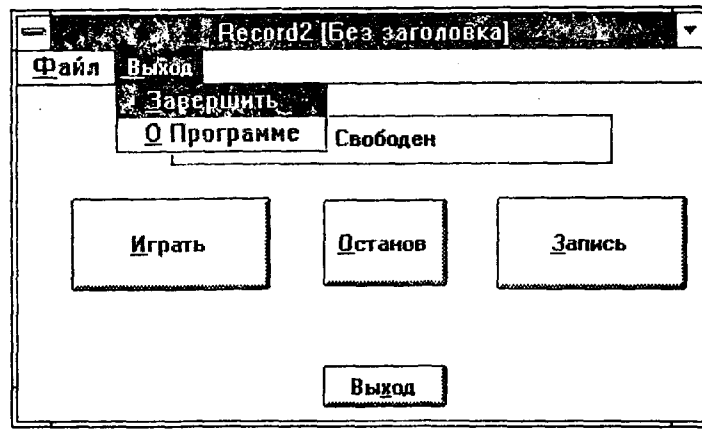


Рис. 11.16. Меню Выход главного окна программы Record2

Меню **Выход** содержит две команды — **Завершить** и **О Программе**. Когда пользователь выбирает команду **Завершить**, программа завершается. Если он выбирает команду **О Программе**, то появляется диалоговое окно, приведенное на рис. 11.17.

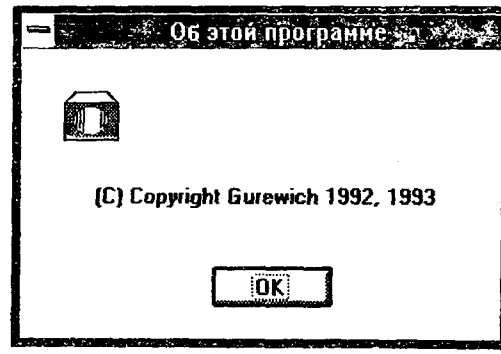


Рис. 11.17. Диалоговое окно команды О Программе программы Record2

Меню **Файл** содержит четыре команды — **Открыть**, **Создать**, **Сохранить** и **Сохранить как...**, которые представляют собой стандартные команды открытия и сохранения файла.

Главное окно программы Record2 содержит четыре командные кнопки — **Играть**, **Останов**, **Запись** и **Выход**. Для записи Wave-файла нажмите командную кнопку **Запись** и произносите в микрофон нужный текст. Для остановки записи нажмите командную кнопку **Останов**, а для воспроизведения записанного файла — кнопку **Играть**. Для закрытия главного окна нажмите командную кнопку **Выход**.

Использовать мышь во время воспроизведения разрешено, и вы можете переключаться на другие приложения Windows.

Программа Record2 позволяет сохранить записанный Wave-файл посредством выбора команд **Сохранить** или **Сохранить как**. Кроме того, Wave-файл можно загрузить, используя команду **Открыть**.

Файлы программы Record2

Программа Record2 состоит из следующих файлов:

Record2.h	#include-файл программы
Record2.c	Текст программы на C
Record2.rc	Файл ресурсов программы
Record2.def	Файл определения модуля программы
Record2.mak	Make-файл программы
ExtSpkr.ico	Пиктограмма программы

Эти файлы записаны на вашем винчестере в директории c:\spSDK\Samp4Win\.

Вариант WM_COMMAND программы Record2

Программа Record2 очень похожа на программу Record; отличие состоит лишь в том, что Record2 может загружать и сохранять файлы с расширением .WAV.

Команда Создать меню Файл

Если пользователь выбирает из меню **Файл** команду **Создать**, то программе передается сообщение WM_COMMAND с wParam, равным IDM_NEW. В варианте IDM_NEW выполняется функция ts_OpenWaveSession() с нулевой строкой, заданной вместо имени файла:

```
lstrcpy( gsWaveFileName, "");
iOpenResult =
ts_OpenWaveSession ( gsWaveFileName );
```

Команда Открыть меню Файл

Если пользователь выбирает из меню **Файл** команду **Открыть**, то программе посылается сообщение WM_COMMAND с wParam, равным IDM_OPEN. В варианте IDM_OPEN мы даем возможность пользователю выбрать файл .WAV, используя стандартное диалоговое окно Windows **Открыть**, изображенное на рис. 11.18. Для этого применяется функция ts_SelectWaveDialog(), которая возвращает ненулевое целое, равное 0, если пользователь не выбрал файл. Если же он выбрал файл, то второй параметр вызова функции ts_SelectWaveDialog() заменяется именем и путем выбранного файла, а третий параметр — именем выбранного файла (без пути).

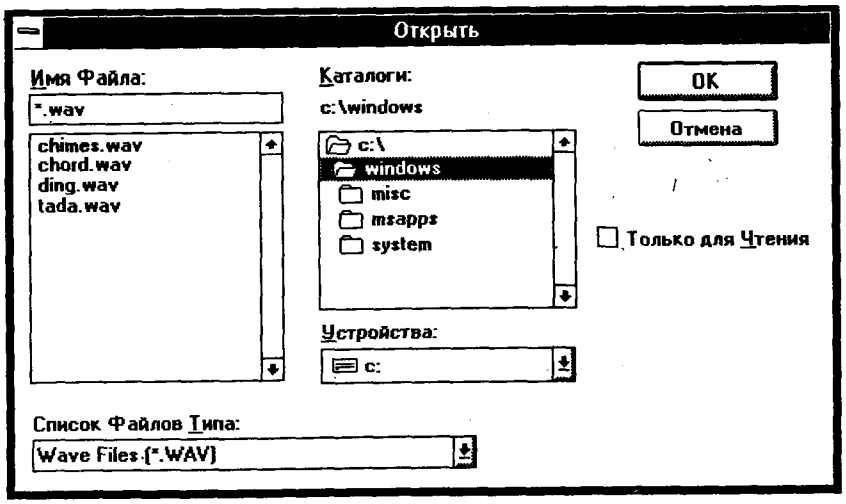


Рис. 11.18. Стандартное диалоговое окно Windows Открыть

Примечание:

Имя Функции: ts_SelectWaveDialog()

Описание: Эта функция дает возможность выбрать файл .WAV, используя стандартное диалоговое окно Windows **Открыть**.

Прототип: int ts_SelectWaveDialog (HWND hWnd,
LPSTR lpstrFileName,
LPSTR lpstrTitle);

Параметры: HWND hWnd: Указатель на окно, в котором появится диалоговое окно **Открыть**.

LPSTR lpstrFileName: Строка, которая будет заменена именем и путем файла, выбранного пользователем.

LPSTR lpstrTitle: Строка, которая будет заменена именем (без пути) файла, выбранного пользователем.

Возвращаемое значение: Целое. Значение 0 указывает на то, что пользователь не выбрал файл — например если он нажал командную кнопку **Отмена** диалогового окна **Открыть**.

После того как пользователь открыл файл, с этим файлом открывается звуковой сеанс:

```
/*-------------------------
Открываем сеанс мультимедиа
с выбранным файлом Wave.
-------------------------*/
iOpenResult =
ts_OpenWaveSession( sSelectedPathAndFile );
```

Если при открытии звукового сеанс, с выбранным файлом произошел сбой, то повторно открываем сеанс с текущим файлом Wave:

```
if ( iOpenResult != 1 )
   {
   lstrcpy ( sMessage,
             "Невозможно открыть сеанс с ");
   lstrcat ( sMessage, sSelectedPathAndFile );
   MessageBox ( NULL,
                sMessage,
                "Сообщение",
                MB_ICONINFORMATION |
                MB_SYSTEMMODAL);

   /*---------------------------------
   Так как невозможно открыть сеанс
   с выбранным файлом, открываем сеанс
   с предыдущим файлом.
   ---------------------------------*/
   iOpenResult =
   ts_OpenWaveSession ( gsWaveFileName );
   DisplayWindowTitle ( hWnd );
   return 0L;
   }
```

Если открытие звукового сеанса прошло успешно, то значение переменной gsWaveFileName устанавливается равным имени выбранного Wave-файла. Переменная gsWaveFileName на протяжении всей работы программы содержит имя Wave-файла, открытого в данный момент. Для отображения этого файла в заголовке окна Record2 используется функция DisplayWindowTitle():

```
lstrcpy ( gsWaveFileName,
          sSelectedPathAndFile );

DisplayWindowTitle ( hWnd );
```

Команда Сохранить меню Файл

Если пользователь выбирает из меню **Файл** команду **Сохранить**, то программе посылается сообщение WM_COMMAND с wParam, равным IDM_SAVE. В варианте IDM_SAVE мы вначале проверяем, является ли имя текущего Wave-файла нулевой строкой. Если имя текущего файла Wave — нулевая строка, то, значит, пользователь еще не сохранил этот файл. Тогда мы обрабатываем этот вариант как вариант **Сохранить как...** посылкой сообщения IDM_SAVE_AS функцией SendMessage(). Это приведет к тому же эффекту, что и выбор пользователем команды меню **Сохранить как**.

Однако если значение переменной gsWaveFileName ненулевое, то результат звукового сеанса сохраняется как файл с расширением .WAV. Для этого используется функция ts_SaveWave():

```
case IDM_SAVE:
  if ( gsWaveFileName[0] == '\0' )
     {
     SendMessage ( hWnd,
                   WM_COMMAND,
                   IDM_SAVE_AS,
                   0);
     return 0L;
     }
  ts_SaveWave ( gsWaveFileName );

  return 0L;
```

Примечание:

Имя Функции: ts_SaveWave()

Описание: Эта функция сохраняет результат звукового сеанса как Wave-файл.

Прототип: int ts_SaveWave (LPSTR sWaveFileName);

Параметр: LPSTR sWaveFileName. Путь и имя файла .WAV, который будет сохраняться.

Возвращаемое значение: Целое. Значение 1 указывает на то, что результат звукового сеанса был успешно сохранен как файл с расширением .WAV; значение, отличное от 1, — на то, что результат звукового сеанса не был сохранен.

Команда Сохранить как... меню Файл

Если пользователь выбирает из меню **Файл** команду **Сохранить как...**, то программе посылается сообщение WM_COMMAND с wParam, равным IDM_SAVE_AS.

В варианте IDM_SAVE_AS мы отображаем для пользователя диалоговое окно **Сохранить как**, где он должен указать имя и директорию файла .WAV, который будет сохранен. Для этого используется функция ts_SaveAsWaveDialog(). Если пользователь закрывает диалоговое окно **Сохранить как** командной кнопкой **Отмена**, то, значит, условие if(!SaveAsResult) выполнено, и вариант завершается оператором return 0L. Если пользователь выбрал файл из диалогового окна **Сохранить как**, то выполняется функция ts_SaveWave(), сохраняющая результат звукового сеанса Wave как файл с расширением .WAV:

```
case IDM_SAVE_AS:
     lstrcpy ( sSelectedPathAndFile, "");
     lstrcpy ( sSelectedFile, "");

     iSaveAsResult =
     ts_SaveAsWaveDialog ( hWnd,
                           sSelectedPathAndFile,
                           sSelectedFile);

     if ( !iSaveAsResult )
        {
        /*-----------------------------------------------
        Пользователь закрыл диалоговое окно Сохранить как.
        -----------------------------------------------*/
        return 0L;
        }

     lstrcpy ( gsWaveFileName,
               sSelectedPathAndFile );

     DisplayWindowTitle ( hWnd );

     ts_SaveWave ( gsWaveFileName );

     return 0L;
```

Примечание:

Имя Функции: ts_SaveAsDialog()

Описание: Эта функция позволяет пользователю выбрать файл с расширением .WAV в стандартном диалоговом окне Windows **Сохранить как**.

Прототип: int ts_SaveAsDialog (HWND hWnd,
LPSTR lpstrFileName,
LPSTR lpstrTitle);

Параметры:

HWND hWnd:	Указатель на окно, в котором появится диалоговое окно **Сохранить как.**
LPSTR lpstrFileName:	Строка, в которой будут помещены имя и путь файла, выбранного пользователем.
LPSTR lpstrTitle:	Строка, в которой будет помещено имя (без пути) файла, выбранного пользователем.

Возвращаемое значение: Целое. Значение 0 указывает на то, что пользователь не выбрал файл — например, нажал командную кнопку **Отмена** диалогового окна **Сохранить как**.

Программа Dog2

В предыдущей главе мы рассматривали программу Dog, которая отображает собаку и ее кость. Когда пользователь пытается "тронуть" кость (щелкнув на ней мышью или нажав клавишу [B]), собака начинает лаять. В программе Dog лай воспроизводится через встроенный динамик ПК. Сейчас мы перепишем программу Dog так, чтобы воспроизводить лай с помощью звуковой платы. Новую программу назовем Dog2.

Программа Dog2 использует несколько новых ts_-функций, которые очень полезны при написании анимационных программ.

Компиляция, компоновка и выполнение программы Dog2

Для компиляции и компоновки программы компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
NMAKE Dog2.mak    [Enter]
```

Для компиляции и компоновки программы компилятором фирмы Borland необходимо:

- Войти в директорию c:\spSDK\Samp4Win.
- После приглашения DOS ввести:

```
MAKE -f Dog2.bmk    [Enter]
```

Перед выполнением программы Dog2 убедитесь в том, что на вашем ПК установлены Windows-совместимая звуковая плата и все необходимые драйверы.

Для выполнения программы Dog2:

- Выберите из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- В диалоговом окне **Пролистать** выберите файл

```
c:\spSDK\Samp4Win\Dog2.exe
```

Появится главное окно программы Dog2, изображенное на рис. 11.19. Меню программы Dog2 содержит три команды: **Завершить**, **О Программе** и **Инструкции** (см. рис. 11.20).

Диалоговое окно **О Программе** программы Dog2 приведено на рис. 11.21, а диалоговое окно **Инструкции** — на рис. 11.22.

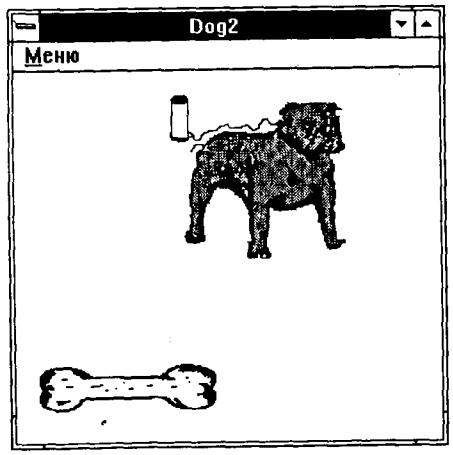


Рис. 11.19. Главное окно Dog2

Рис. 11.20. Меню программы Dog2

Рис. 11.21. Диалоговое окно О Программе Dog2

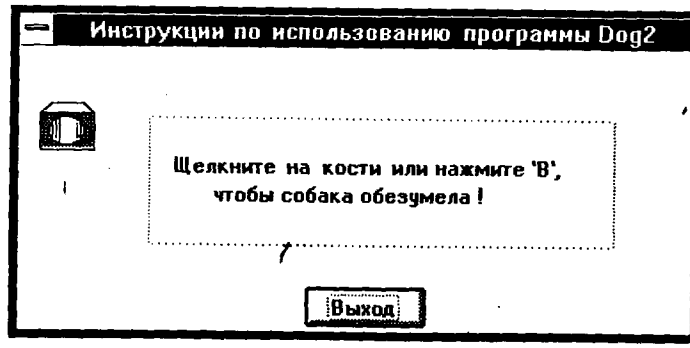


Рис. 11.22. Диалоговое окно Инструкции по использованию программы Dog2

Когда пользователь щелкает мышью на изображении кости (или нажимает клавишу [B]), собака начинает прыгать и лаять. Лай воспроизводится с помощью звуковой платы.

Файлы программы Dog2

Программа Dog2 состоит из следующих файлов:

Dog2.h	#include-файл программы
Dog2.c	Текст программы на C
Dog2.rc	Файл ресурсов программы
Dog2.def	Файл определения модуля программы
Dog2.mak	Make-файл программы
ExtSpkr.ico	Пиктограмма программы

Эти файлы записаны на вашем винчестере в директории c:\spSDK\Samp4Win\.

Анимация в программе Dog2

Существуют два типа анимации: *асинхронная* и *синхронная*.

В асинхронной анимации воспроизводимый звук не синхронизируется с движущимся изображением. Пример асинхронной анимации — программа Dog2.

В синхронной анимации воспроизводимый звук синхронизирован с движущимся изображением, причем каждый фрейм связан со своим собственным звуковым фрагментом. Далее в этой главе будет рассмотрена программа, в которой используется синхронная анимация.

Процесс создания асинхронной демонстрации

В этом разделе описан процесс создания асинхронной демонстрации, который включает в себя установку таймера.

Подготовка таймера

Перед началом асинхронной демонстрации необходимо установить таймер. В программе Dog2 таймер устанавливается в функции WinMain():

```
/*--------------------
Устанавливаем таймер.
---------------------*/
if ( !SetTimer ( hWnd,
     1,
     175,   /* Каждые 175 мс. */
     NULL ))
   {
   MessageBox ( NULL,
                "Больше нет свободных таймеров !",
                "Проблемы с таймером !",
                MB_ICONINFORMATION );

   /*--------------------
   Завершаем приложение.
   ---------------------*/
   SendMessage ( hWnd,
                 WM_DESTROY,
                 0,
                 0 );

   }/* конец if() */
```

Оператор SetTimer() "заставляет" оболочку Windows каждые 175 мс генерировать сообщение WM_TIMER. Значение установки таймера (175 мс) определяет частоту, с которой в процессе демонстрации сменяются фреймы. Мы используем значение 175 мс, а вы можете поэкспериментировать и с другими значениями.

Подготовка фреймов для демонстрации

Демонстрация состоит из нескольких фреймов (файлов с расширением .BMP). Для их создания применялась функция ts_EditFrame().

В варианте WM_PAINT выполняется функция PrepareShow1(). Эта функция создает демонстрацию номер 1, которая содержит два рисунка: кость и собаку с закрытой пастью. Ниже показано, как в функции PrepareShow1() создается растровое изображение кости:

```
ts_EditFrame ( 0,     /* Номер фрейма.          */
               ghInst,/* Экземпляр приложения. */
               ghBone,/* Указатель растрового изображения.     */
               10,    /* Левая верхняя X-координата в пикселах.*/
               200,   /* Левая верхняя Y-координата в пикселах.*/
               125,   /* Ширина в пикселах.     */
               32,    /* Высота в пикселах.     */
               -1L,   /* Начальная звуковая координата: Нет */
               -1L ); /* Конечная звуковая координата: Нет  */
```

Подобным же образом в функции PrepareShow1() создается и растровый рисунок собаки:

```
ts_EditFrame ( 1,       /* Номер фрейма.          */
               ghInst,  /* Экземпляр приложения.*/
               ghDogWithCloseMouth, /* Указатель растрового */
                        /*  изображения.                    */
               100,     /* Левая верхняя X-координата в пикселах.*/
               10,      /* Левая верхняя Y-координата в пикселах.*/
               125,     /* Ширина в пикселах.  */
               125,     /* Высота в пикселах.  */
               -1L,     /* Начальная звуковая координата: Нет */
               -1L );   /* Конечная звуковая координата: Нет  */
```

Последнее, что делает PrepareShow1() — это маркирует фрейм #1 как последний фрейм демонстрации, для чего применяется функция ts_MarkLastFrame():

```
/*-----------------------------
 Помечаем фрейм #1 как последний.
 -----------------------------*/
 ts_MarkLastFrame (1);
```

Примечание:

Имя Функции: ts_EditFrame()

Описание: Эта функция создает растровое изображение для демонстрации.

Прототип: int ts_EditFrame (int iFrameNumber,
HINSTANCE hInstance,
HBITMAP hBitMap,
int iX,
int iY,
int iWidth,
int iHeight,
long lStartSND,
long lEndSND);

Параметры:

int iFrameNumber:	Номер фрейма.
HINSTANCE hInstance:	Экземпляр программы.
HBITMAP hBitMap:	Указатель растрового изображения.
int iX:	X-координата фрейма в пикселах.
int iY:	Y-координата фрейма в пикселах.
int iWidth:	Ширина фрейма в пикселах.
int iHeight:	Высота фрейма в пикселах.
longlStartSND:	В асинхронной анимации не используется.
long lEndSND:	В асинхронной анимации не используется.

Возвращаемое значение: Эта функция всегда возвращает 1.

Примечание:

Имя Функции: ts_MarkLastFrame()

Описание: Эта функция маркирует фрейм как последний фрейм демонстрации.

Прототип: int ts_MarkLastFrame (int iFrameNumber);

Параметры: int iFrameNumber, номер фрейма, который будет маркирован как последний фрейм демонстрации.

Возвращаемое значение: Эта функция всегда возвращает 1.

После того как демонстрация подготовлена, для отображения отдельных ее фреймов может быть использована функция ts_DisplayFrozenFrame(). В варианте WM_PAINT мы отображаем изображения кости и собаки:

```
/*-----------
Рисуем кость.
-------------*/
ts_DisplayFrozenFrame ( hWnd, 0 );
```

```
/*-----------------------------
Рисуем собаку с закрытой пастью.
-----------------------------*/
ts_DisplayFrozenFrame ( hWnd, 1 );
```

Примечание:
Имя Функции: ts_DisplayFrozenFrame()

Описание: Эта функция отображает один фрейм.

Прототип: int ts_DisplayFrozenFrame (HWND hWnd,
int iFrameNumber);

Параметры:
HWND hWnd: Указатель окна, в котором будет отображен фрейм.
int iFrameNumber: Номер фрейма, который будет отображен.

Возвращаемое значение: Эта функция всегда возвращает 1.

Демонстрация номер 1 готова; теперь можем подготовить демонстрацию номер 2, для чего применим функцию PrepareShow2().

Функция PrepareShow2()

Демонстрация номер 2 состоит из двух фреймов. Первый — изображение собаки с открытой пастью, второй — с закрытой (таким образом, функция PrepareShow2() вызывает функцию ts_EditFrame() дважды). Функция PrepareShow2() с помощью функции ts_MarkLastFrame() маркирует второй фрейм как последний фрейм демонстрации:

```
void PrepareShow2 ( void )
{
/*-------------------------------------
Строим фрейм #0: Собака с закрытой пастью
-------------------------------------*/
ts_EditFrame (0,        /* Номер фрейма.          */
              ghInst,   /* Экземпляр приложения. */
              ghDogWithCloseMouth, /* Указатель растрового */
                        /* изображения.                  */
              100,      /* Левая верхняя X-координата в пикселах.*/
              10,       /* Левая верхняя Y-координата в пикселах.*/
              125,      /* Ширина в пикселах.   */
              125,      /* Высота в пикселах.   */
              -1L,      /* Начальная звуковая координата: Нет  */
              -1L );    /* Конечная звуковая координата: Нет   */

/*-------------------------------------
Строим фрейм #1: Собака с открытой пастью
-------------------------------------*/
ts_EditFrame (1,        /* Номер фрейма.          */
              ghInst,   /* Экземпляр приложения. */
              ghDogWithOpenMouth,  /* Указатель растрового */
                        /* изображения.                  */
              100,      /* Левая верхняя X-координата в пикселах.*/
              10,       /* Левая верхняя Y-координата в пикселах.*/
              125,      /* Ширина в пикселах.   */
              125,      /* Высота в пикселах.   */
              -1L,      /* Начальная звуковая координата: Нет */
              -1L );    /* Конечная звуковая координата: Нет  */
```

```
/*-------------------------------
Маркируем фрейм #1 как последний.
-------------------------------*/
ts_MarkLastFrame (1);

}/* Конец функции. */
```

Воспроизведение звука в процессе демонстрации

Демонстрация номер 2 включает в себя воспроизведение звука с помощью звуковой платы. Воспроизведение начинается после того, как пользователь нажимает клавишу [B] или щелкает мышью на изображении кости.

Когда пользователь нажимает клавишу [B], начинается воспроизведение звука и глобальная переменная giTheShowMustGoOn устанавливается в 1.

```
case 'B':
  /*-----------------------
  Готовимся отобразить лай.
  -----------------------*/
  PrepareShow2();

  /*-----------------------
  Воспроизводим лай собаки.
  -----------------------*/
  if ( ts_PlayWave ( hWnd,
                     TS_START_OF_FILE,
                     TS_END_OF_FILE,
                     TS_IN_MILLISECONDS ) == 1)
     {
     giTheShowMustGoOn = 1;
     }
  return 0;
```

Анимация

Помните о том, что таймер установлен в 175 мс. Это означает, что функция WndProc() принимает сообщение WM_TIMER каждые 175 мс. В варианте WM_TIMER проверяется переменная giTheShowMustGoOn. Если флаг установлен в 1, то выполняется функция ts_ShowAsync():

```
case WM_TIMER:
     if ( giTheShowMustGoOn == 1 )
        {
        ts_ShowAsync ( hWnd );
        }
     return 0L;
```

Примечание:

Имя Функции: ts_ShowAsync();

Описание: Эта функция отображает следующий фрейм демонстрации. После того как отображен последний фрейм (фрейм, маркированный как последний), следующим отображается первый фрейм демонстрации.

Прототип: int ts_ShowAsync (HWND hWnd);

Параметр: HWND hWnd: Указатель окна, в котором будет отображен фрейм.

Возвращаемое значение: Эта функция всегда возвращает 1.

Завершение демонстрации

Завершаем демонстрацию после приема сообщения MM_MCINOTIFY. Как вы уже знаете, сообщение MM_MCINOTIFY с параметром wParam, равным MCI_NOTIFY_SUCCESSFUL, принимается после того, как звуковая плата завершит воспроизведение. В варианте MCI_NOTIFY_SUCCESSFUL демонстрация завершается установкой флага giTheShowMustGoOn в 0. Затем посредством подготовки и показа фреймов демонстрации номер 1 отображаются собака с закрытой пастью и ее кость:

```
case MM_MCINOTIFY:

    switch ( wParam )
           {
           case MCI_NOTIFY_SUCCESSFUL:
                giTheShowMustGoOn = 0 ;
                /*-------------------------------------
                Демонстрация завершилась, оставим собаку
                с закрытой пастью.
                -------------------------------------*/
                PrepareShow1();
                ts_DisplayFrozenFrame ( hWnd, 0 );
                ts_DisplayFrozenFrame ( hWnd, 1 );
                break;

           case MCI_NOTIFY_ABORTED:
                ts_StopWave();
                giTheShowMustGoOn = 0;
                break;

           case MCI_NOTIFY_SUPERSEDED:
                ts_StopWave();
                giTheShowMustGoOn = 0;
                break;

           case MCI_NOTIFY_FAILURE:
                ts_StopWave();
                giTheShowMustGoOn = 0;
                break;

           }/* конец switch() */

    return 0L;
```

Программа PressAny

А теперь мы приступим к созданию программы PressAny, которая представляет собой пример синхронной анимации. Здесь в анимационной демонстрации отображаемый текст "Press any key to continue..." будет синхронизирован со звуковой фразой *Press any key to continue...*, которая воспроизводится с помощью звуковой платы.

Компиляция, компоновка и выполнение программы PressAny

Для компиляции и компоновки программы компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win.

➤ После приглашения DOS введите:

```
NMAKE PressAny.mak    [Enter]
```

Для компиляции и компоновки программы компилятором фирмы Borland необходимо:

➤ Войти в директорию c:\spSDK\Samp4Win.

➤ После приглашения DOS ввести:

```
MAKE -f PressAny.bmk    [Enter]
```

Перед выполнением программы PressAny убедитесь в том, что на вашем ПК установлены Windows-совместимая звуковая плата и все необходимые драйверы.

Для выполнения программы PressAny:

➤ Выберите из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.

➤ В диалоговом окне **Пролистать** выберите файл

```
c:\spSDK\Samp4Win\PressAny.exe
```

Появится главное окно программы PressAny, изображенное на рис. 11.23.

Меню программы PressAny содержит три команды: **Завершить**, **О Программе** и **Инструкции** (см. рис. 11.24). Диалоговое окно **О Программе** программы PressAny представлено на рис. 11.25, а диалоговое окно **Инструкции** — на рис. 11.26.

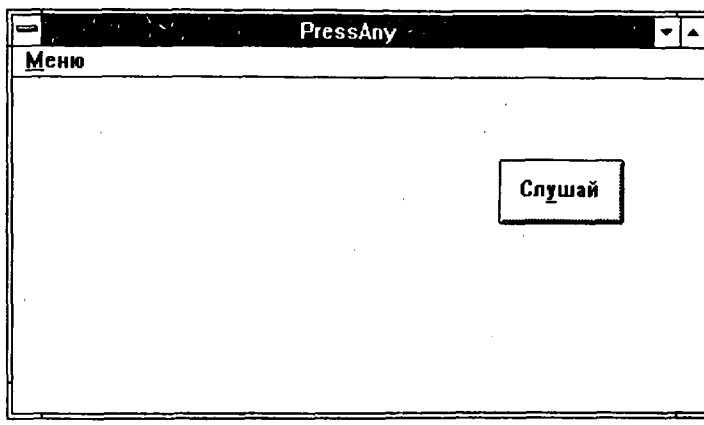


Рис. 11.23. Главное окно программы PressAny

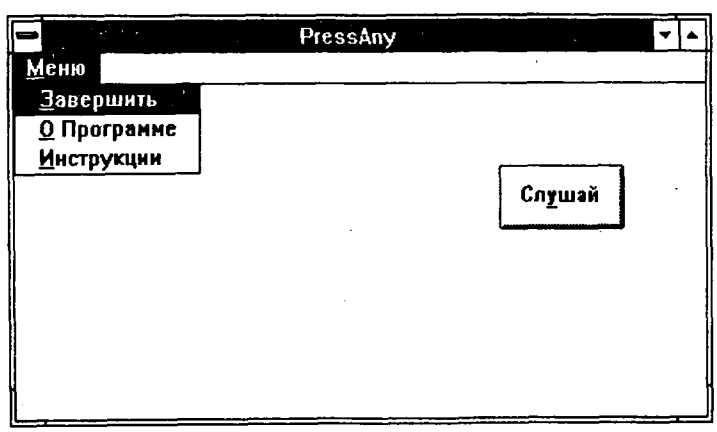


Рис. 11.24. Меню программы PressAny

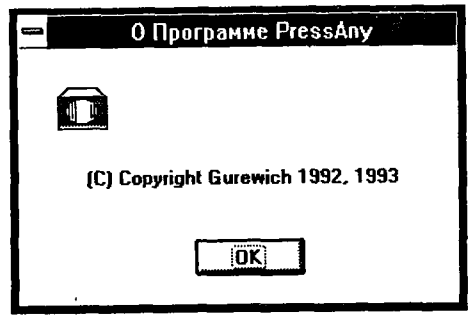


Рис. 11.25. Диалоговое окно О Программе PressAny

Когда пользователь нажимает командную кнопку **Слушай** (или клавишу [y] на клавиатуре компьютера), отображается текст "Press any key to continue" (см. рис. 11.27) и одновременно (синхронно) воспроизводится звуковая фраза *Press any key to continue....*

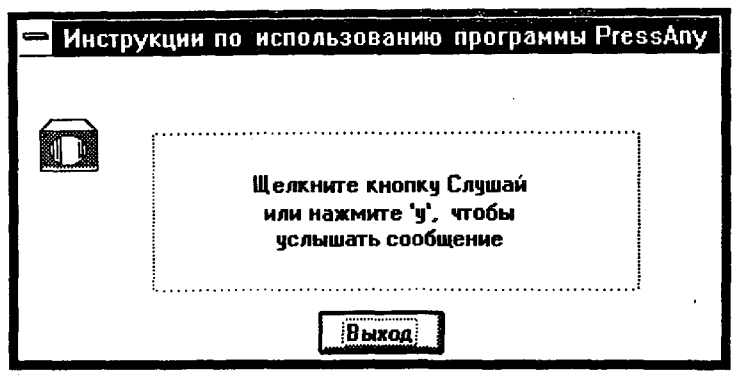


Рис. 11.26. Диалоговое окно Инструкции по использованию программы PressAny

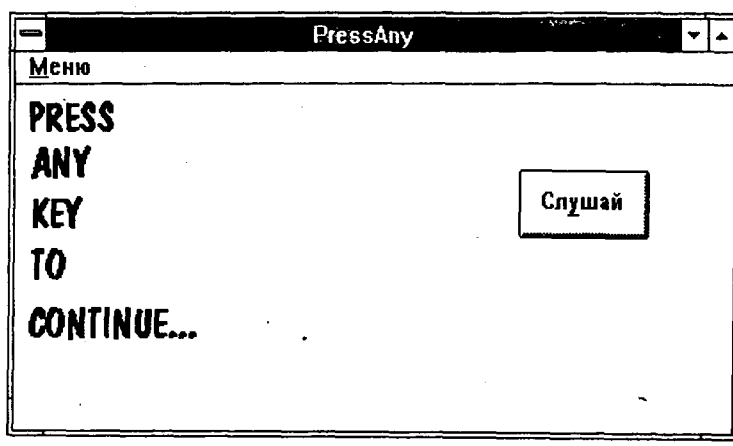


Рис. 11.27. Отображенный текст "Press any key to continue..."

Файлы программы PressAny

Программа PressAny состоит из следующих файлов:

PressAny.h	#include-файл программы
PressAny.c	Текст программы на C
PressAny.rc	Файл ресурсов программы
PressAny.def	Файл определения модуля программы
PressAny.mak	Make-файл программы
ExtSpkr.ico	Пиктограмма программы

Эти файлы записаны на вашем винчестере в директории c:\spSDK\Samp4Win\.

Программа PressAny использует растровые изображения Wait6.bmp, Wait7.bmp, Wait8.bmp, Wait9.bmp и Wait10.bmp (приведены на рис. 11.28); указанные файлы расположены в директории c:\spSDK\Samp4Win\.

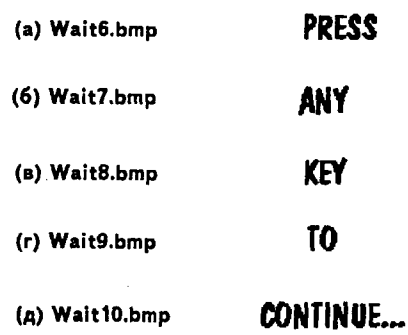


Рис. 11.28. Растровые изображения, используемые в программе PressAny

Демонстрация в программе PressAny

Программа PressAny очень похожа на программу Dog2; отличие состоит лишь в том, что демонстрация PressAny синхронизирована с воспроизведением фрагментов звукового файла (синхронная демонстрация).

Подготовка таймера

Таймер устанавливается в функции WinMain() так, чтобы сообщения WM_TIMER генерировались каждые 75 мс:

```
/*--------------------
Устанавливаем таймер.
---------------------*/
if ( !SetTimer ( hWnd,
                 1,
                 75,   /* Каждые 75 мс */
                 NULL ))
   {
   MessageBox ( NULL,
                "Больше нет свободных таймеров !",
                "Проблемы с таймером !",
                MB_ICONINFORMATION );
```

```
/*--------------------
 Завершаем приложение.
 --------------------*/
 SendMessage ( hWnd,
               WM_DESTROY,
               0,
               0 );

 }/* конец if() */
```

Подготовка демонстрации

Демонстрация создается с помощью функции PrepareShow(), которая пять раз вызывает функцию ts_EditFrame() (для пяти фреймов), а затем, используя функцию ts_MarkLastFrame(), помечает пятый фрейм как последний.

Поскольку демонстрация синхронизирована с воспроизведением звука, то последние два параметра функции ts_EditFrame() представляют диапазон *звукового захвата*. Отображенный на экране фрейм должен находиться там до тех пор, пока будет воспроизводиться указанный фрагмент звукового файла.

Приведем функцию ts_EditFrame() для фрейма номер 0:

```
/*------------------------
 Строим фрейм #0: "PRESS"
 ------------------------*/
ts_EditFrame ( 0,       /* Номер фрейма.          */
               ghInst,  /* Экземпляр приложения. */
               ghPress,/* Указатель растрового изображения.   */
               10,     /* Левая верхняя X-координата в пикселах.*/
               10,     /* Левая верхняя Y-координата в пикселах.*/
               60,      /* Ширина в пикселах. */
               25,      /* Высота в пикселах. */
               0L,/*    /* Начальная звуковая координата. */
               800L ); /* Конечная звуковая координата.  */
```

В данном случае при вызове функции ts_EditFrame() указывается диапазон от 0 до 800 мс (два последних параметра функции). Это означает, что растровое изображение, заданное указателем ghPress, должно отобразиться на экране в тот момент, когда будет воспроизводиться звуковой фрагмент, находящийся в указанном диапазоне.

Подобным же образом с помощью функции ts_EditFrame() устанавливаются звуковые захваты для других фреймов. Ниже приведены значения звуковых захватов для пяти фреймов нашей программы:

Номер фрейма	*Указатель растрового изображения*	*Звуковой диапазон (в миллисекундах)*	*Воспроизводимое слово*
0	ghPress	0 – 800	Press
1	ghAny	800 – 1100	any
2	ghKey	1100 – 1600	key
3	ghTo	1600 – 2080	to
4	ghContinue	2080 – 2780	continue

Начало демонстрации

Если пользователь нажимает командную кнопку **Слушай**, то программе посылается сообщение WM_COMMAND с wParam, равным HEARME_PB. Демонстрация в варианте HEARME_PB создается с помощью функции PrepareShow(), а затем запускается функцией воспроизведения звукового файла ts_PlayWave(). Если значение, возвращаемое функцией ts_PlayWave(), равно 1, то звуковая плата приняла команду воспроизведения, и в этом варианте флаг goTheShowMustGoOn установлен в 1:

```
case HEARME_PB:
     /*-------------------
     Готовим демонстрацию.
     ---------------------*/
     PrepareShow();

     /*---------------------
     Выполняем демонстрацию.
     -----------------------*/
     if ( ts_PlayWave ( hWnd,
                        TS_START_OF_FILE,
                        TS_END_OF_FILE,
                        TS_IN_MILLISECONDS ) == 1)

        {
        giTheShowMustGoOn = 1;
        }
    return 0L;
```

Переменная giTheShowMustGoOn используется как флаг для разрешения или запрета демонстрации в варианте WM_TIMER.

Сообщение WM_TIMER

Как уже упоминалось, функция SetTimer() "заставляет" систему каждые 75 мс генерировать сообщение WM_TIMER. Синхронизация демонстрации выполняется в варианте WM_TIMER:

```
case WM_TIMER:
  if ( giTheShowMustGoOn == 1 )
     {
     ts_ShowSync ( hWnd );
     }
  return 0L;
```

Функция ts_ShowSync() отображает фрейм, соответствующий воспроизводимому в данный момент фрагменту звукового файла. Напомним, что с помощью функции ts_EditFrame() были определены фреймы для каждого звукового диапазона. Таким образом, функция ts_ShowSync() проверяет, в каком диапазоне находится воспроизводимый байт, и, основываясь на этом значении, отображает соответствующий фрейм.

Примечание:

Имя Функции: ts_ShowSync()

Описание: Эта функция отображает фрейм, соответствующий воспроизводимому фрагменту звукового файла.

Прототип: int ts_ShowSync (HWND hWnd);

Параметр: HWND hWnd: Указатель окна, в котором будет отображаться фрейм.

Возвращаемое значение: Всегда равно 1.

Для того чтобы не пропустить момент смены фреймов, установка таймера должна иметь достаточно малое значение. В данном случае мы задаем таймеру установку 75 мс. Это значение достаточно мало и позволяет отобразить все фреймы демонстрации.

Завершение демонстрации

Мы завершаем демонстрацию после приема сообщения MM_MCINOTIFY. Напомним, что сообщение MM_MCINOTIFY с параметром wParam, равным MCI_NOTIFY_SUCCESSFUL, принимается после того, как звуковая плата завершит воспроизведение. В варианте MCI_NOTIFY_SUCCESSFUL демонстрация завершается установкой флага giTheShowMustGoOn в 0. Функции InvalidateRect() и UpdateWindow() используются для генерации сообщения WM_PAINT и очистки окна.

```
case MM_MCINOTIFY:

     switch ( wParam )
            {
            case MCI_NOTIFY_SUCCESSFUL:
                 InvalidateRect ( hWnd, NULL, TRUE );
                 UpdateWindow ( hWnd );
                 giTheShowMustGoOn = 0;
                 break;

            case MCI_NOTIFY_ABORTED:
                 InvalidateRect ( hWnd, NULL, TRUE );
                 UpdateWindow ( hWnd );
                 giTheShowMustGoOn = 0;
                 ts_StopWave();
                 break;

            case MCI_NOTIFY_SUPERSEDED:
                 InvalidateRect ( hWnd, NULL, TRUE );
                 UpdateWindow ( hWnd );
                 giTheShowMustGoOn = 0;
                 ts_StopWave();
                 break;

            case MCI_NOTIFY_FAILURE:
                 InvalidateRect ( hWnd, NULL, TRUE );
                 UpdateWindow ( hWnd );
                 giTheShowMustGoOn = 0;
                 ts_StopWave();
                 break;

            }/* конец switch() */

  return 0L;
```

ts_-функции анимации

ts_-функции анимации, применяемые в этой программе, значительно облегчают жизнь программистов. Используя эти высокоуровневые функции, вы можете сконцентрировать свое внимание на художественных аспектах демонстрации, экспериментировать с ней, не утруждая себя при этом написанием повторяющихся фрагментов текста

программы. Для того чтобы расширить демонстрацию за счет добавления новых фреймов, достаточно вставить в функцию PrepareShow() дополнительные вызовы функции ts_EditFrame() для определения этих фреймов.

К сожалению, сокращенная версия библиотеки TS содержит ограниченное количество ts_-функций анимации.

Воспроизведение файлов MIDI

Файлы MIDI (цифровой интерфейс музыкальных инструментов) широко применяются в музыкальных программах. Они имеют небольшой объем и могут быть использованы для фонового музыкального сопровождения.

Сейчас мы рассмотрим программу PlayMIDI, которая воспроизводит файлы MIDI.

Файлы программы PlayMIDI

Программа PlayMIDI состоит из следующих файлов:

PlayMIDI.h	#include-файл программы
PlayMIDI.c	Текст программы на C
PlayMIDI.rc	Файл ресурсов программы
PlayMIDI.def	Файл определения модуля программы
PlayMIDI.mak	Make-файл программы
PlayMIDI.ico	Пиктограмма программы

Эти файлы доступны для вас в директории c:\spSDK\Samp4Win\.

Компиляция, компоновка и выполнение программы PlayMIDI

Для компиляции и компоновки программы PlayMIDI компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
NMAKE PlayMIDI.mak    [Enter]
```

Для компиляции и компоновки программы компилятором фирмы Borland выполните следующие действия:

- Войдите в директорию c:\spSDK\Samp4Win.
- После приглашения DOS введите:

```
MAKE -f PlayMIDI.bmk  [Enter]
```

Перед выполнением программы PlayMIDI убедитесь в том, что на вашем ПК установлены Windows-совместимая звуковая плата, способная воспроизводить файлы MIDI (например Sound Blaster), а также все необходимые драйверы.

Для выполнения программы PlayMIDI:

- Выберите из меню **Файл** Диспетчера Программ Windows команду **Выполнить**.
- В диалоговом окне **Пролистать** выберите файл

```
c:\spSDK\Samp4Win\PlayMIDI.exe
```

Выберите из меню команду **Открыть** и откройте произвольный MIDI-файл. Файлы MIDI обычно имеют расширение .MID. Директория Windows, как правило, содержит пример файла MIDI (c:\Windows\Canyon.mid).

Теперь вы можете воспроизвести выбранный MIDI-файл нажатием командной кнопки **Играть**.

Текст программы PlayMIDI

Программа PlayMIDI очень похожа на программу Record2. Различие между ними состоит лишь в том, что PlayMIDI не имеет командной кнопки **Запись**, а также в том, что функция ts_Wave_ была заменена на функцию ts_MIDI_, функция ts_WaveDeviceCanPlay() — на функцию ts_MIDIDeviceCanPlay(), функция ts_SelectWaveDialog() — на ts_SelectMIDIDialog(), ts_OpenWaveSession() — на ts_OpenMIDISession() , ts_PlayWave() — на ts_PlayMIDI(), ts_StopWave() — на ts_StopMIDI(), а ts_CloseWaveSession() — на ts_CloseMIDISession().

Другие функции ts_MIDI

Библиотека TS, поставляемая на дискете вместе с этой книгой, представляет собой сокращенную версию и, следовательно, содержит ограниченное количество функций ts_MIDI_. Полная же версия библиотеки содержит множество дополнительных функций ts_MIDI, позволяющих увеличивать и уменьшать громкость воспроизведения файлов MIDI, записывать такие файлы, получать текущую позицию воспроизводимого файла (для звуковых захватов) и выполнять другие полезные действия.

Микширование файлов .WAV и MIDI

Многие программы воспроизводят файлы MIDI одновременно с файлами .WAV. Вы можете запустить программу PlayMIDI, запустить воспроизведение файла MIDI, а затем, пока этот файл будет воспроизводиться, переключиться в программу Dog2 и воспроизвести лай. Вы услышите музыку MIDI, звучащую одновременно с лаем собаки. Естественно, вы можете воспроизвести файлы .WAV и MIDI из одной программы. Эту возможность можно использовать при необходимости добавить фоновую музыку (файл MIDI) к вокалу (файл .WAV).

Глава 12. Программирование звукового сопровождения на C для DOS

В предыдущих главах рассматривалось программирование звукового сопровождения на языке C для Windows. Сейчас вы ознакомитесь со способами создания звуковых программ на C для DOS.

Из данной главы вы узнаете, как воспроизводить звуковые файлы через встроенный динамик персонального компьютера, а из следующей — как применять для этих целей звуковую плату Sound Blaster.

Создание программ для воспроизведения звуковых файлов через встроенный динамик ПК

Если вы — программист, получающий удовольствие от работы в среде Windows, то у вас может возникнуть вполне закономерный вопрос: зачем тратить время на изучение программирования в среде DOS, которая значительно уступает Windows по своим возможностям? Оставим этот вопрос пока без ответа и перейдем прямо к делу — к созданию C-программ для воспроизведения звуковых файлов в среде DOS.

Поскольку мы создаем программу для DOS, она должна быть переносимой, т.е. не должна требовать применения каких-либо специальных драйверов или любого другого специально устанавливаемого программного обеспечения. Это достигается за счет использования библиотеки TegoMS.lib. Сокращенная версия этой библиотеки, поставляемая на прилагаемой к данной книге дискете, включает в себя достаточное количество функций для того, чтобы можно было компилировать и компоновать все представленные в книге программы.

Файл MAKEexe.bat

В директории c:\spSDK\SampMS\ находится командный файл MAKEexe.bat, который содержит командную строку cl для компиляции и компоновки C-программ компилятором Microsoft C. Приведем листинг этого файла:

```
echo off
cls
Echo ************************************************************
Echo *                                                          *
Echo *Компиляция и компоновка компилятором Microsoft C и звуковой*
Echo *   библиотекой для DOS TegoSoft C (сокращенная версия).    *
Echo *                                                          *
Echo ************************************************************
cl -AH  %1   C:\spSDK\Lib\TegoMS.lib C:\c700\lib\Graphics.lib
```

В командной строке cl указаны две библиотеки:

1. c:\spSDK\Lib\TegoMS.lib
2. c:\c700\lib\Graphics.lib

Первая представляет собой сокращенную версию библиотеки TegoMS.lib. Она предназначена для применения совместно с компилятором Microsoft C.

Вторая — Graphics.lib — поставляется вместе с компилятором Microsoft C. Если в вашей программе не используются функции из Graphics.lib, то эту библиотеку можно смело удалить из файла MAKEexe.bat.

В директории c:\spSDK\SampBL\ находится аналогичный файл — MAKEexe.bat, содержащий командную строку bcc для компиляции и компоновки С-программ компилятором Borland C. Приведем листинг файла c:\spSDK\SampBL\MAKEexe.bat:

```
echo off
cls
Echo ************************************************************
Echo *                                                          *
Echo * Компиляция и компоновка компилятором Borland C и звуковой *
Echo *   библиотекой для DOS TegoSoft C (сокращенная версия).   *
Echo *                                                          *
Echo ************************************************************
bcc -mh %1  c:\spSDK\Lib\TegoBL.lib
```

Командная строка для bcc содержит ссылку на библиотеку c:\spSDK\Lib\TegoBL.lib, которая является сокращенной версией библиотеки TegoBL.lib. Эта библиотека предназначена для применения совместно с компилятором Borland C.

Программа PlayTS.c

Программа PlayTS позволяет пользователю воспроизводить любой звуковой файл типа TS из командной строки DOS. Воспроизведение звуковых файлов других типов (например, файлов с расширением .WAV, файлов с расширением .VOC) мы рассмотрим позже.

Генерация звуковых файлов типа TS

Звуковой файл типа TS может быть сгенерирован из Wave-файла с помощью утилиты WAV2TS.exe.

Пример: *Преобразование звукового файла с расширением .WAV в звуковой файл с расширением .TS*

Утилита WAV2TS.exe преобразует любой Wave-файл в звуковой файл типа TS.

Например, для преобразования звукового файла Hello.wav в звуковой файл типа TS вы должны применить утилиту WAV2TS.exe следующим образом:

После приглашения DOS введите:

```
WAV2TS  Hello.wav Hello.ts     [Enter]
```

Результирующим файлом будет звуковой файл Hello.ts.

Обратите внимание на то, что утилиты WAV2TS.exe нет на дискете, прилагаемой к данной книге. Эта утилита входит в состав редактора Sound Editor фирмы TS. Авторы исходили из предположения, что читатель не располагает этим редактором, поэтому вначале был создан файл Hello.wav, а затем использована утилита WAV2TS.exe для создания файла Hello.ts. Файл Hello.ts расположен в директории c:\spSDK\TSfiles\.

Рекомендуем перед изучением текста программы PlayTS откомпилировать, скомпоновать и выполнить данную программу — это позволит лучше понять ее возможности.

Компиляция и компоновка программы PlayTS

Для компиляции и компоновки программы компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.

- Войдите в директорию c:\spSDK\SampMS\.
- После приглашения DOS введите:

```
MAKEexe PlayTS.c    [Enter]
```

Кроме того, для компиляции и компоновки файла PlayTS.c можно использовать программу PWB, которая поставляется вместе с компилятором Microsoft C. При открытии проекта с помощью PWB не забудьте подключить библиотеки c:\spSDK\Lib\TegoMS.lib и c:\c700\lib\Graphics.lib.

Для компиляции и компоновки программы компилятором фирмы Borland необходимо:

- Войти в директорию c:\spSDK\SampBL\.
- После приглашения DOS ввести:

```
MAKEexe PlayTS.c    [Enter]
```

Выполнение программы PlayTS.exe

Программа PlayTS позволяет воспроизводить любой звуковой файл типа TS через встроенный динамик персонального компьютера.

Для воспроизведения файла c:\spSDK\TSfiles\Hello.ts следует выполнить следующие действия:

- Убедитесь в том, что вы работаете в DOS (если вы перешли в режим DOS из Windows, то нужно завершить сеанс Windows).
- Войдите в директорию c:\spSDK\SampMS\.
- После приглашения DOS введите:

```
PlayTS c:\spSDK\TSfiles\Hello.ts    [Enter]
```

Результатом будет воспроизведение звукового файла Hello.ts в бесконечном цикле. Остановить этот процесс можно нажатием любой клавиши.

Подобным же образом можно воспроизвести любой файл типа TS, расположенный в директории c:\spSDK\TSfiles\.

Примечание:

Если вы откомпилировали и скомпоновали программу компилятором фирмы Borland, то вы должны поступить следующим образом:

Войти в директорию c:\spSDK\SampBL\.

После приглашения DOS ввести:

```
PlayTS c:\spSDK\TSfiles\Hello.ts    [Enter]
```

Как выбирать имена для звуковых файлов

Все звуковые файлы типа TS, поставляемые на дискете данной книги, имеют расширение .ts. Вообще же звуковые файлы могут иметь любые расширения, допустимые в DOS. Это позволяет давать им произвольные, удобные для вас и отражающие их назначение имена, например Play.Me, Hear.Me, Fun2Hear и т.п.

Текст программы PlayTS

Текст программы PlayTS находится в файле c:\spSDK\SampMS\PlayTS.c. Ниже приведен листинг данной программы.

Листинг 12.1. PlayTS.c

```
/*============================================================
 ИМЯ ПРОГРАММЫ: PlayTS.c

 (C) Copyright Gurewich 1992, 1993 (R) All Rights Reserved.

 ОПИСАНИЕ ПРОГРАММЫ:
 -------------------
 Эта программа воспроизводит звуковой файл .TS от начала до
 конца в бесконечном цикле.

 Пользователь может остановить воспроизведение нажатием
 любой клавиши.
 ============================================================*/

/*========
  #include
 =========*/
/*-----------------------------
 Стандартные #include-файлы C.
 -----------------------------*/
#include <conio.h>
#include <graph.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>

/*-----------------------------------------
 Стандартные #include-файлы TegoSoft Sound+
 (сокращенный вариант).
 -----------------------------------------*/
#include "c:\spsdk\h\sp1.h"
#include "c:\spsdk\h\sp2.h"
#include "c:\spsdk\h\sp3.h"

/*========
 прототипы
 =========*/
void display_status_line ( char * message_s );

/*==============
 ФУНКЦИЯ: main()
 ===============*/
void main ( int argc, char *argv[ ] )
{
/*--- Имя звукового файла, который будет воспроизводиться. ---*/
char  file_name_to_play_s[125];
/*--- Результат открытия звукового сеанса. ---*/
int   open_result_i;

/*-----------------------------
 Последний воспроизведенный байт
-------------------------------*/
long   current_byte_l;
```

```
/*-------------
 Очистим экран.
 -------------*/
_clearscreen (_GCLEARSCREEN );

/*-------------------------------------------------
 Получим имя звукового файла из командной строки DOS.
 -------------------------------------------------*/
strcpy ( file_name_to_play_s, argv[1] );

/*-------------------------------
 Поручим TSEngine открыть сеанс.
 -------------------------------*/
display_status_line ("Открытие сеанса. Пожалуйста, подождите...");
open_result_i =
sp_open_tsengine_session ( file_name_to_play_s, SP_TS_FILE );
if ( open_result_i != 1 )
   {
   printf ( "\n Невозможно открыть сеанс для звукового файла:%s",argv[1]);
   exit(0);
   }

/*-------------------------------------
 Ожидаем, пока пользователь нажмет клавишу
 для запуска воспроизведения.
 -------------------------------------*/
display_status_line ("Нажмите любую клавишу для начала воспроизведения...");
SP_CLEAR_TYPEAHEAD;
getch();

display_status_line ( "Идет воспроизведение, нажмите любую клавишу для
                       прерывания воспроизведения..." );
current_byte_l = 0L;

_disable();
while ( 1 )
      {
      current_byte_l =
      sp_play_byte_range ( current_byte_l, current_byte_l + 15000L );

      /*-----------------------------------------------------------
      Если звуковой файл полностью воспроизведен, начнем все сначала.
      -----------------------------------------------------------*/
      if (current_byte_l == -1L )
         {
         current_byte_l = 0L;
         }
      /*-----------------------------------
      Если пользователь нажмет любую клавишу,
      завершим воспроизведение.
      -----------------------------------*/
      if ( kbhit() )
         break;
      } /* конец while(1) */
_enable();
```

```
display_status_line ( "До свидания" );
_settextposition ( 24, 1 );

}/* конец main() */
/*=== конец main() ===*/

/*========
  ФУНКЦИЯ
  ========*/
/*----------------------------------------
 ОПИСАНИЕ:
 Функция удаляет строку номер 22, а затем
 отображает сообщение по центру этой строки.
 Эта функция использована для отображения
 состояния и сообщений-инструкций.
 ----------------------------------------*/
void display_status_line ( char * message_s )
{
int message_length_i;  /* Длина отображаемого сообщения.    */
int i;                 /* Использовано в качестве счетчика.*/

/*-------------------------------------------
 Определим длину сообщения (для центрирования).
 -------------------------------------------*/
message_length_i = strlen ( message_s );

/*------------------------
 Очищаем строку состояния.
 ------------------------*/
_settextposition ( 22, 1 );
for (i=0;i<79;i++)
    printf(" ");

/*-------------------------------------
 Отображаем сообщение по центру строки.
 -------------------------------------*/
_settextposition ( 22, 40-message_length_i/2 );
printf("%s", message_s);

}
/*=== Конец функции. ===*/
```

#include-файлы

Существуют две группы #include-файлов — стандартные #include-файлы C (например, conio.h, graph.h) и стандартные #include-файлы TegoSoft Sound+ — sp1.h, sp2.h и sp3.h. Эти файлы расположены в директории c:\spSDK\h\.

Примечание:

Чтобы обеспечить звуковое сопровождение программы на C для DOS, нужно включить в нее #include-файлы sp1.h, sp2.h и sp3.h, которые содержат макросы и прототипы C-функций для работы со звуком.

Получение имени звукового файла из командной строки DOS

Функции main() передаются параметры argc и *argv[]:

```
void main ( int argc, char argv[ ] )
{
.....
.....
.....
}
```

Путь и имя звукового файла копируются в строковую переменную file_name_to_play_s:

```
strcpy ( file_name_to_play_s, argv[1] );
```

В С-программах, приведенных в данной книге, авторы придерживались следующего соглашения об именах переменных: имя каждой переменной заканчивается буквой, указывающей на ее тип. Так, строковая переменная file_name_to_play_s завершается символом s, длинная целая переменная current_byte_l — символом l и т.п. Разумеется, при создании собственных программ вы не обязаны следовать этому соглашению.

Открытие звукового сеанса

Перед воспроизведением звукового файла следует открыть звуковой сеанс. Для этой цели применяется функция sp_open_tsengine_session():

```
open_result_i =
sp_open_tsengine_session ( file_name_to_play_s, SP_TS_FILE );
if ( open_result_i != 1 )
   {
   printf ( "\n Невозможно открыть сеанс для звукового файла:%s",argv[1]);
   exit(0);
   }
```

Функция sp_open_tsengine_session() включена в сокращенную версию библиотеки c:\spSDK\Lib\TegoMS.lib. Это можно определить по имени функции: оно начинается с sp_.

Примечание:

Имена всех функций из библиотеки c:\spSDK\Lib\TegoMS.lib (или c:\spSDK\Lib\TegoBL.lib) начинаются с символов sp_, имена всех макросов и определений — с символов SP_.

Имя функции: sp_open_tsengine_session()

Описание: Эта функция открывает звуковой сеанс. Прототип *sp_open_tsengine_session()* объявлен в файле c:\spSDK\h\sp1.h как

```
int sp_open_tsengine_session ( char * file_name_to_play_s,
                               int file_type_i );
```

Параметры:

char * file_name_to_play_s:	Строка, оканчивающаяся нулем, которая содержит путь и имя звукового файла, подлежащего открытию.
int file_type_i:	Тип звукового файла, который будет открыт.

Примеры правильных значений типов файла:

SP_S_FILE	Звуковые файлы типа S.
SP_TS_FILE	Звуковые файлы типа TS.

Использование функций, включенных в полную версию библиотеки, позволяет открывать звуковые сеансы с файлами типа SP_WAV_FILE (для файлов .WAV), SP_VOC_FILE (для файлов .VOC) и SP_SND_FILE (для файлов .SND).

Применение идентификаторов такого типа обеспечивает возможность воспроизводить файлы с расширениями .WAV, .VOC и звуковые файлы других распространенных типов без использования утилиты для их преобразования к нужному типу. Например, когда в качестве второго параметра функции sp_open_tsengine_session() указывается SP_WAV_TYPE, будет открыт звуковой файл типа .WAV. Таким образом, отпадает необходимость в утилите WAV2TS.exe.

Возвращаемое значение: Целое, которое содержит результат открытия звукового сеанса. Возвращаемое значение, отличное от 1, означает, что TSEngine не смогла открыть звуковой сеанс.

Очистка буфера клавиатуры

После открытия звукового файла программа приглашает пользователя нажать любую клавишу для запуска воспроизведения. Но предварительно следует очистить буфер клавиатуры, чтобы иметь уверенность в том, что в буфере клавиатуры не остался помещенный туда ранее код символа, который программа могла бы интерпретировать как результат нажатия клавиши пользователем. Поскольку такая очистка в программах на C, использующих звуковое сопровождение, осуществляется довольно часто, то в файле sp2.h определен макрос SP_CLEAR_TYPEAHEAD:

```
#define SP_CLEAR_TYPEAHEAD  while (kbhit()) \
                                getch()
```

Воспроизведение звукового файла

Звуковой файл воспроизводится с помощью функции sp_play_byte_range().

Примечание:

Имя функции: sp_play_byte_range()

Описание: Эта функция воспроизводит звуковой файл (или фрагмент звукового файла).

Прототип: long sp_play_byte_range (long req_start_l, long req_end_l);

Параметры:

long req_start_l: Первый байт фрагмента звукового файла, который будет воспроизведен.
long req_end_l: Последний байт фрагмента звукового файла, который будет воспроизведен.

Возвращаемое значение: Длинное целое, представляющее собой последний воспроизведенный байт. Отрицательная величина означает, что фрагмент звукового файла не был воспроизведен.

Цикл воспроизведения

Воспроизведение выполняется в цикле while(1) следующим образом:

```
while ( 1 )
      {
      current_byte_l =
```

```
sp_play_byte_range ( current_byte_l, current_byte_l + 15000L );

/*---------------------------------------
Если звуковой файл полностью воспроизведен,
начнем все сначала.
---------------------------------------*/
if (current_byte_l == -1L )
   {
   current_byte_l = 0L;
   }
/*-------------------------------------
Если пользователь нажмет любую клавишу,
завершим воспроизведение.
-------------------------------------*/
if ( kbhit() )
   break;
} /* конец while(1) */
```

Звуковой файл воспроизводится фрагментами по 15000 байтов каждый. На самой первой итерации цикла воспроизводится фрагмент звукового файла от current_byte_l = 0L до байта со смещением 15000. После завершения воспроизведения этого фрагмента возвращаемое функцией sp_play_byte_range() значение равно 15000L. На следующей итерации цикла воспроизводимый фрагмент расположен от current_byte_l = 15000L до байта со смещением 30000. После воспроизведения каждого фрагмента проверяется значение переменной current_byte_l. Процесс воспроизведения фрагментами продолжается до тех пор, пока второй параметр функции sp_play_byte_range() не превысит размер звукового файла.

Так, если размер звукового файла — 35000 байтов и на первой итерации был воспроизведен фрагмент от 0 до 15000 байтов, а на второй — от 15000 до 30000, то на третьей итерации будет воспроизведен фрагмент от 30000 до 35000 байтов. Действительно, на третьей итерации следовало воспроизвести фрагмент от 30000 до 45000 байтов, но поскольку общий размер звукового файла в данном примере — только 35000 байтов, то TSEngine воспроизведет лишь фрагмент от байта со смещением 30000 до байта со смещением 35000, после чего возвратит значение –1L, означающее, что достигнут конец звукового файла.

Как уже упоминалось, в теле цикла while(1) значение, возвращаемое функцией sp_play_byte_range(), проверяется на –1. Если это значение равно -1, то, значит, звуковой файл был воспроизведен полностью, и в этом случае переменная current_byte_l устанавливается в 0L. Но можно модифицировать программу так, чтобы в случае возврата значения –1 осуществлялся выход из цикла while(1) и, таким образом, звуковой файл воспроизводился только один раз.

Прерывание цикла while(1)

Возможность прервать цикл while(1) нажатием любой клавиши в данной программе реализована с помощью оператора if(kbhit()). Он используется для проверки, была ли нажата клавиша на клавиатуре компьютера. Если условие if(kbhit()) удовлетворено, т.е. пользователь нажал клавишу, выполняется оператор break и происходит выход из цикла:

```
if ( kbhit() )
   break;
```

Стандартные функции C _disable() и _enable()

Функции _disable() и _enable() — это стандартные функции C, которые служат для запрета и разрешения прерываний. Если перед выполнением функции sp_play_byte_range() выполняется функция _disable(), то качество звука будет выше. Для того чтобы разрешить прерывания, в конце воспроизведения следует использовать функцию _enable().

Запрещать прерывания во время воспроизведения необязательно (такого требования нет). Но это делать целесообразно, поскольку таким образом, как уже упоминалось, можно значительно улучшить качество звука. Однако необходимо учитывать следующее: если ваша программа использует мышь, то после выполнения функции _disable() мышь будет запрещена и снова станет активной только после выполнения функции _enable(). Но поскольку использование мыши при воспроизведении может привести к нежелательным звуковым эффектам, то и с этой точки зрения применение функции _disable(), безусловно, является полезным.

Другие функции семейства sp_play_

Функция sp_play_byte_range() относится к семейству функций sp_play_. Как видно из ее имени, координаты фрагмента звукового файла, который будет воспроизведен этой функцией, должны быть указаны в байтах. Кроме sp_play_byte_range() семейство sp_play включает в себя функции sp_play_fsec_range() и sp_play_label_range().

Функция sp_play_fsec_range() подобна функции sp_play_byte_range(). Различие между ними состоит лишь в том, что координаты фрагмента звукового файла, который будет воспроизведен этой функцией, должны быть указаны в секундах. Например, для воспроизведения звукового фрагмента от 1,32 до 4,32 с следует задать функцию в виде

```
sp_play_fsec_range ( 1.32, 4.32 );
```

В функции sp_play_label_range() координаты фрагмента звукового файла, который будет воспроизведен этой функцией, должны быть указаны в виде звуковых меток. Звуковые метки предварительно должны быть помещены в файл. Для этой цели используется редактор Sound Editor фирмы TS.

Рассмотрим типичный пример применения звуковых меток.

Предположим, что в процессе работы над программой вы создали звуковой файл, в котором с помощью Sound Editor было объединено несколько файлов. Такой файл может включать в себя фразы *"Good-Bye", "Welcome", "Press any key to continue...", "Having fun?"* и другие музыкальные или речевые фрагменты. Далее вы можете поставить метку *"Press* начинается здесь" в начале звукового фрагмента *Press any key to continue...*, а метку *"Press* заканчивается здесь" — в конце этого фрагмента. Аналогично можно поставить звуковые метки в начале и конце других звуковых фрагментов этого файла.

Для воспроизведения конкретного звукового фрагмента в программе следует использовать функцию sp_play_label_range(). Так, для воспроизведения фразы *"Press any key to continue..."* необходимо выполнить функцию с такими параметрами:

```
sp_play_label_range ( "Press начинается здесь",
                      "Press заканчивается здесь");
```

Поскольку Sound Editor не включен в дискету, прилагаемую к данной книге, и авторы, как уже неоднократно упоминалось, исходили из предположения, что читатель не располагает этим редактором, то во всех представленных в книге примерах программ используется функция sp_play_byte_range().

Управление памятью и виртуальная память

Одним из наиболее замечательных свойств библиотеки c:\spSDK\Lib\TegoMS.lib является то, что включенные в нее функции полностью реализуют все необходимые операции обмена с диском и распределение памяти, так что пользователю остается лишь указать координаты фрагмента звукового файла, который следует воспроизвести. Так, предположим, что у вас есть звуковой файл объемом 10 Мбайт. Независимо от того, каков объем ОЗУ вашего компьютера, для воспроизведения любого фрагмента этого файла можно уверенно применять функцию sp_play_byte_range(). Если объема ОЗУ недостаточно для того, чтобы хранить 10 Мбайт звукового файла, то функция sp_play_byte_range() будет автоматически производить все необходимые операции подкачки, выгрузки и загрузки независимо от указанного диапазона воспроизведения.

Программа PlayS

Программа PlayS похожа на программу PlayTS, но с ее помощью можно задавать воспроизведение файлов типа S непосредственно в командной строке DOS.

Компиляция, компоновка и выполнение программы PlayS

Для компиляции и компоновки файла c:\spSDK\SampMS\PlsayS.c компилятором фирмы Microsoft выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\SampMS\.
- После приглашения DOS введите:

```
MAKEexe PlayS.c [Enter]
```

Для компиляции и компоновки файла c:\spSDK\SampMS\PlsayS.c компилятором фирмы Borland необходимо:

- Войти в директорию c:\spSDK\SampBL\.
- После приглашения DOS ввести:

```
MAKEexe PlayS.c [Enter]
```

Для выполнения программы PlayS:

- Если вы компилировали компилятором фирмы Microsoft, то войдите в директорию c:\spSDK\SampMS\.
- Если вы использовали компилятор фирмы Borland, то войдите в директорию c:\spSDK\SampBL\.
- После приглашения DOS введите:

```
PlayS c:\spSDK\Sfiles\Press.s [Enter]
```

Программу PlayS можно использовать для воспроизведения любых звуковых файлов типа S, расположенных в директории c:\spSDK\Sfiles.

Текст программы PlayS

Файл PlayS.c почти идентичен файлу PlayTS.c. Различие между ними состоит лишь в том, что второй параметр функции sp_open_tsengine_session() равен SP_S_FILE:

```
sp_open_tsengine_session ( file_name_to_play_s, SP_S_FILE );
```

Использование файлов .S

Если сравнить файлы из директорий c:\spSDK\TSfiles\ и c:\spSDK\Sfiles\, то легко заметить, что файлы с расширением .S меньше по размеру, чем аналогичные файлы с расширением .TS.

Звуковые файлы типа S могут быть получены из файлов с расширением .WAV с помощью утилиты WAV2S.exe.

Пример:

Преобразование звукового файла типа WAV в звуковой файл типа S

Утилита WAV2S.exe преобразует любой файл типа WAV в звуковой файл типа S.

Так, для преобразования файла Press.wav в звуковой файл типа S нужно следующим образом применить утилиту WAV2S.exe:
После приглашения DOS введите:

```
WAV2S Press.wav Press.s [Enter]
```

В результате вы получите звуковой файл Press.s.

Утилита WAV2S.exe не включена в дискету, прилагаемую к данной книге. Эта утилита входит в состав редактора Sound Editor фирмы TS. А поскольку авторы исходят из предположения, что читатель не располагает Sound Editor, то вначале был создан файл Press.wav, а затем использована утилита WAV2S.exe для создания файла Press.s (он расположен в директории c:\spSDK\Sfiles\).

Объем воспроизводимого фрагмента

Цикл while(1) программы PlayS.c воспроизводит файл типа S фрагментами по 15000 байтов каждый.

Используя формулу преобразования для файлов типа S

$$\text{Длительность в секундах} = \frac{\text{Размер в байтах} \times 8}{\text{Частота дискретизации в Гц}},$$

можно убедиться, что каждые 15000 байтов звукового файла Press.s (записанного с частотой дискретизации 40000 Гц) воспроизводятся в течение трех секунд:

$$\text{Длительность в секундах} = \frac{15000 \times 8}{40000} = 3 \text{ с}.$$

Размер воспроизводимого фрагмента определяет скорость реакции программы на нажатие клавиши. В программе PlayS пользователь может ожидать реакции на нажатие клавиши до трех секунд, и лишь после этого воспроизведение будет прервано. Чтобы уменьшить время реакции, нужно уменьшить размер воспроизводимых фрагментов звукового файла.

Программа PlayTS воспроизводит файлы типа TS. Воспользовавшись формулой, приведенной ниже, можно по заданному размеру звукового файла (в байтах) определить длительность его звучания (в секундах).

$$\text{Длительность в секундах} = \frac{\text{Размер в байтах}}{\text{Частота дискретизации в Гц}}.$$

Так, воспроизведение 15000 байтов файла типа TS, записанного с частотой 8000 Гц, занимает

$$\text{Длительность в секундах} = \frac{15000}{8000} = 1{,}875 \text{ с}.$$

Таким образом, при работе с программой PlayTS после нажатия клавиши может произойти задержка до 1,875 с, прежде чем воспроизведение будет прервано.

Автономные программы

До сих пор все рассматриваемые нами программы на С, разрабатываемые для DOS, были неавтономными звуковыми программами. Это означает, что дистрибутивная дискета каждой из них (дискета, содержащая все файлы данной программы) должна содержать не только исполняемый файл, но и все звуковые файлы, используемые программой. Предположим, что вы хотите переслать кому-либо файл, содержащий звуковое сообщение *"Hello. Have a nice day. Good-Bye"*. В этом случае ваш дистрибутивная дискета должна содержать два файла — PlayTS.exe и Hello.ts.

Когда программа состоит из более чем одного файла, это не всегда удобно для пользователя: всегда есть риск случайно стереть один из файлов или поместить их в разные директории.

Чтобы избежать этого, можно преобразовать неавтономную программу в автономную. В этом случае, для того чтобы передать кому-либо программу, достаточно будет записать на дистрибутивную дискету только один автономный файл HearMe.exe. Для получения сообщения, содержащегося в этом файле, пользователь просто вводит **HearMe** после приглашения DOS.

В данном случае звуковой файл входит в состав файла HearMe.exe.

Преобразование программы PlayTS в автономную программу

Чтобы поупражняться, давайте преобразуем неавтономную программу PlayTS, которая рассматривалась ранее, в автономную. Для этого выполним следующие действия:

ШАГ 1:

Скопируем файл PlayTS.c в TSalone.c:

```
COPY playTS.c TSalone.c [Enter]
```

Теперь внесем изменения в текст программы TSalone.c.

ШАГ 2:

Заменим функцию sp_open_tsengine_session() функцией sp_open_tsengine_session_sa():

```
sp_open_tsengine_session_sa ( argv[0], 0L, SP_TS_FILE );
```

Слово "sa" в имени функции обозначает *Stand-Alone* (автономная).

Примечание:

Имя функции: sp_open_tsengine_session_sa()

Описание: Эта функция используется для открытия звукового сеанса в автономной программе. Прототип sp_open_tsengine_sa() объявлен в файле c:\spSDK\h\sp1.h как
int sp_open_tsengine_session_sa(char * file_name_to_play_s,
long not_applicable_l, int file_type_i);

Параметры:
char * file_name_to_play_s: Строка, заканчивающаяся нулем, которая содержит путь и имя открываемого звукового файла. Для автономных программ этот параметр должен быть равен argv[0] (то есть путь и имя автономного EXE-файла).

long not_applicable_l: Этот параметр в сокращенной версии библиотеки не используется. В качестве его значения можно указать 0L.

int file_type_i: Этим параметром задается тип звукового файла, подлежащего открытию.

Примеры правильных значений данного параметра:
SP_S_FILE Звуковые файлы типа S
SP_TS_FILE Звуковые файлы типа TS

Для полной версии библиотеки допустимы значения SP_WAV_FILE (для файлов .WAV), SP_VOC_FILE (для файлов типа VOC) и SP_SND_FILE (для файлов типа SND).

Эти идентификаторы обеспечивают возможность воспроизводить файлы .WAV, .VOC и звуковые файлы других распространенных типов, не применяя утилиту преобразования. Так, если в качестве третьего параметра функции sp_open_tsengine_session() используется значение SP_WAV_TYPE, открываемым звуковым файлом будет файл .WAV и, следовательно, нет необходимости применять утилиту WAV2S.exe.

Возвращаемое значение: Целое. Если возвращаемое значение равно 1, то библиотека TSEngine не смогла открыть звуковой сеанс, в противном случае открытие прошло успешно.

Для преобразования неавтономной звуковой программы в автономную в тексте исходной С-программы необходимо только заменить вызовы функции sp_open_tsengine_session() вызовами функции sp_open_tsengine_session_sa().

ШАГ 3:

На этом шаге компилируем и компонуем автономную программу TSalone.c:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\SampMS\ — для компилятора фирмы Microsoft или в директорию c:\spSDK\SampBL\ — для компилятора фирмы Borland
- После приглашения DOS введите:

```
MAKEexe TSAlone.c [Enter]
```

ШАГ 4:

Последний шаг — это объединение файла TSalone.exe (сгенерированного на шаге 3) со звуковым файлом c:\spSDK\TSfiles\Hello.ts.

Утилита TSlink

Утилита TSlink расположена в директории c:\spSDK\Util\. Эта утилита позволяет объединить в один файл EXE-файл и звуковой файл типа S или TS.

Утилита TSlink, содержащаяся на дискете этой книги, представляет собой сокращенную версию. Полная ее версия, которая позволяет подключать звуковые файлы различных типов, например WAV и VOC, включена как составная часть в редактор Sound Editor фирмы TS.

Для использования утилиты TSlink

- Войдите в директорию c:\spSDK\SampMS\ — для компилятора фирмы Microsoft или в директорию c:\spSDK\SampBL\ — для компилятора фирмы Borland.

➤ После приглашения DOS введите:

```
c:\spSDK\Util\TSlink TSalone.exe c:\spSDK\TSFiles\Hello.ts NiceDay.exe [Enter]
```

Первый параметр утилиты TSlink — это имя компонуемого исполняемого файла, второй — имя компонуемого звукового файла, который будет объединен с исполняемым; третий параметр — имя результирующего исполняемого файла. Необходимо указывать расширения имен файлов; следовательно, первым параметром должно быть имя TSalone.exe, а не TSalone, вторым — Hello.ts, а не Hello, третьим — NiceDay.exe, а не NiceDay.

Примечание:

Утилита TSlink.exe объединяет исполняемый файл со звуковым файлом.

Синтаксис: TSlink первый_параметр второй_параметр третий_параметр

Первый параметр: Имя компонуемого исполняемого файла.

Второй параметр: Имя компонуемого звукового файла.

Третий параметр: Имя результирующего исполняемого файла.

Выполнение автономной программы NiceDay

Вы можете вызвать автономную программу из командной строки DOS, для чего:

➤ Войдите в директорию c:\spSDK\SampMS\ — для компилятора фирмы Microsoft; или в директорию c:\spSDK\SampBL\ — для компилятора фирмы Borland.

➤ После приглашения DOS введите

```
NiceDay [Enter]
```

Автономная программа представляет собой один файл NiceDay.exe, который может находиться на любом диске и в любой директории.

С помощью утилиты TSlink звуковой файл Hello.ts был объединен с файлом TSalone.exe и создан новый файл с именем NiceDay.exe. Объем файла NiceDay.exe приблизительно равен сумме объемов файлов TSalone.exe и Hello.ts. Теперь дистрибутивная дискета будет содержать только один файл — NiceDay.exe, так что в распространении звукового файла Hello.ts больше нет необходимости.

Преобразование других программ

Описанным выше способом можно преобразовать и все остальные неавтономные звуковые программы, приведенные в этой главе, в автономные.

Дополнительные функции sp_ из звуковой библиотеки TegoSoft

Кроме семейств функций sp_open_ и sp_play_ библиотека TegoMS.lib включает в себя и другие sp_-функции, позволяющие различными способами манипулировать звуковыми файлами и извлекать из них информацию. Следующая программа показывает, как можно извлечь информацию из звуковых файлов.

Программа Info4TS

Файл Info4TS.c записан на вашем винчестере в директории c:\spSDK\SampMS\. Эта программа использует различные функции из семейства функций sp_get_.

Компиляция, компоновка и выполнение программы Info4TS

Для компиляции и компоновки программы Info4TS выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\SampMS\ — для компилятора фирмы Microsoft; или в директорию c:\spSDK\SampBL\ — для компилятора фирмы Borland.
- После приглашения DOS введите:

```
MAKEexe Info4TS.c [Enter]
```

Для выполнения программы Info4TS:

- Войдите в директорию c:\spSDK\SampMS\ — для компилятора фирмы Microsoft; или в директорию c:\spSDK\SampBL\ — для компилятора фирмы Borland.
- После приглашения DOS введите:

```
Info4TS  путь\имя_файла.ts [Enter]
```

- Например, для выполнения программы Info4TS с файлом c:\spSDK\TSfiles\Music.ts введите после приглашения DOS

```
Info4TS c:\spSDK\TSfiles\Music.ts [Enter]
```

Подобным же образом можно выполнить программу Info4TS с любым из других файлов типа TS, находящихся в c:\spSDK\TSfiles\.

После запуска программы Info4TS на экране отображается информация о звуковом файле, после чего выводится сообщение с приглашением нажать любую из клавиш для запуска воспроизведения. Во время воспроизведения на экране отображается текущая позиция в воспроизводимом звуковом файле. Пользователь может прервать воспроизведение нажатием любой клавиши на клавиатуре компьютера.

Текст программы Info4TS.c

Текст программы Info4TS находится в директории c:\spSDK\SampMS\.

Как всегда, раздел #include-файлов содержит стандартные #include-файлы с расширением h библиотеки TegoMS.lib:

```
/*--- Стандартные header-файлы TegoSoft Sound+. ---*/
#include "c:\spsdk\h\sp1.h"
#include "c:\spsdk\h\sp2.h"
#include "c:\spsdk\h\sp3.h"
```

Запрещение прерывания по [Ctrl+C]

Запрещение прерывания по [Ctrl+C] позволяет предотвратить прерывание работы программы по инициативе пользователя. Для этого используется функция signal():

```
/*-------------------------------------------------------
Запретим Ctrl_C плюс другие сигналы так, чтобы пользователь
не мог завершить программу нажатием [Ctrl+C].
-------------------------------------------------------*/
signal(SIGINT,SIG_IGN);
```

Получение информации о файлах типа .TS

После открытия сеанса функцией sp_open_tsengine_session() программа может получить различного рода информацию о звуковом файле.

Объем звукового файла в байтах можно получить с помощью функции sp_get_size_of_file_in_lbytes():

```
/*--- Получим размер звукового файла TS в байтах. ---*/
size_of_file_in_bytes_l = sp_get_size_of_file_in_lbytes();
```

Слово lbytes в имени функции говорит о том, что возвращаемое этой функцией значение является длинным целым, представляющим объем звукового файла в байтах.

Длительность звучания звукового файла в секундах определяется с помощью функции sp_get_size_of_file_in_lsec():

```
/*--- Получим значение длительности звучания ---*/
/*---  звукового файла типа TS в секундах.   ---*/
size_of_file_in_lsec_l = sp_get_size_of_file_in_lsec();
```

Слово lsec в имени функции означает, что возвращаемое данной функцией значение является длинным целым, представляющим длительность звучания звукового файла в секундах.

Кроме того, для получения значения длительности звучания звукового файла в секундах можно воспользоваться функцией sp_get_size_of_file_in_fsec():

```
/*--- Получим значение длительности звучания ---*/
/*---  звукового файла типа TS в секундах.   ---*/
size_of_file_in_fsec_f = sp_get_size_of_file_in_fsec();
```

Выражение fsec в имени функции означает, что возвращаемое данной функцией значение является числом с плавающей точкой, представляющим длительность звучания звукового файла в секундах.

Значение частоты дискретизации можно получить с помощью функции sp_get_sampling_rate():

```
/*--- Частота дискретизации. ---*/
sampling_rate_l = sp_get_sampling_rate();
```

Значение текущей скорости воспроизведения определяется с помощью функции sp_get_playback_speed():

```
/*--- Скорость воспроизведения. ---*/
playback_speed_f = sp_get_playback_speed();
```

Отображение текущей позиции при воспроизведении

Воспроизведение осуществляется при выполнении цикла while(1). Звуковой файл воспроизводится фрагментами по 8000 байтов каждый. В промежутках между воспроизведением фрагментов для извлечения информации о текущей позиции вызывается функция sp_get_posiotion_in_fsec():

```
 disable();
7*--- Воспроизводим файл. ---*/
display_message("Идет воспроизведение...");
current_byte_l = 0L;
while ( 1 )
   {
   current_byte_l =
   sp_play_byte_range ( current_byte_l,
                        current_byte_l + 8000L );
```

```
        /*--- Выход из цикла, если файл TS полностью воспроизведен.---*/
        if ( current_byte_l == -1L )
             break;

        /*-- Завершение воспроизведения, если была нажата клавиша.--*/
        if ( kbhit() )
             break;

        /*--- Значение текущей позиции в секундах. ---*/
        position_in_fsec_f = sp_get_position_in_fsec();
        _settextposition ( row_number_i, 10 );
        printf ( "Текущая позиция TS-файла %.2f секунд.",
                  position_in_fsec_f);
        }
    _enable();
```

Программа Info4S

Программа Info4S практически идентична программе Info4TS. Отличие состоит лишь в том, что программа Info4S работает с файлами типа S.

Компиляция, компоновка и выполнение программы Info4S

Для компиляции и компоновки программы Info4S выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\SampMS\ — для компилятора фирмы Microsoft; или в директорию c:\spSDK\SampBL\ — для компилятора фирмы Borland.
- После приглашения DOS введите:

```
MAKEexe Info4S.c [Enter]
```

Для выполнения программы Info4S:

- Войдите в директорию c:\spSDK\SampMS\ — для компилятора фирмы Microsoft; или в директорию c:\spSDK\SampBL\ — для компилятора фирмы Borland.
- После приглашения DOS введите:

```
Info4S путь\имя_файла.s [Enter]
```

- Так, для выполнения программы Info4S с файлом c:\spSDK\Sfiles\Press.s после приглашения DOS введите:

```
Info4S c:\spSDK\Sfiles\Press.s [Enter]
```

Подобным же образом можно выполнить программу Info4S и с любым другим файлом типа S, находящимся в директории c:\spSDK\Sfiles\.

Применение семейства функций sp_get_ для звуковых файлов других типов (WAV, VOC, SND)

Как можно увидеть, различие между Info4TS и Info4S состоит во втором параметре функции sp_open_tsengine(). После того как звуковой сеанс открыт, все функции sp_ могут быть использованы независимо от типа звукового файла. Таким образом, семейство функций sp_get_ (так же, как и другие sp_-функции) может применяться для файлов типа S, WAV, VOC и звуковых файлов других типов. Обратите внимание на то, что поставляемая на дискете этой книги библиотека представляет собой сокращенную версию, и поэтому при ее использовании возможна работа только со звуковыми файлами типа S и TS.

Управление процессом воспроизведения звукового файла

Иногда при воспроизведении звуковых файлов может возникнуть необходимость вмешательства в процесс воспроизведения. Например, вы создали программу HearMe, воспроизводящую звуковой файл с длительностью звучания 25 минут. Естественно, трудно представить себе пользователя, способного слушать информацию, пусть даже самую интересную, на протяжении всех 25 минут без перерыва. Человека всегда может что-либо отвлечь, например телефонный звонок. Для таких случаев в программе воспроизведения звукового файла целесообразно предусмотреть средства управления процессом воспроизведения, аналогичные тем, которыми оснащен кассетный магнитофон. Пользователь должен иметь возможность остановить на время воспроизведение (**Пауза**), а также начать воспроизведение с указанной позиции (**Перемотка назад** и **Перемотка вперед**). Рекомендуется также обеспечить в программе HearMe возможность изменять скорость воспроизведения — тогда пользователь сможет быстрее просканировать запись в поисках нужной позиции.

Для установки желаемой скорости воспроизведения следует воспользоваться функцией sp_set_playback_speed(). Прототип этой функции объявлен в файле c:\spSDK\h\sp2.h как

```
void sp_set_playback_speed ( float );
```

Так, установка удвоенной (по сравнению с естественной) скорости воспроизведения осуществляется следующим образом:

```
sp_set_playback_speed ( 2.0 );
```

установка половинной (по сравнению с естественной) скорости естественного воспроизведения:

```
sp_set_playback_speed ( 0.5 );
```

установка естественной скорости воспроизведения:

```
sp_set_playback_speed ( 1.0 );
```

Полная версия библиотеки TegoMS.lib включает множество других sp_-функций, которые обеспечивают возможность манипулировать звуковыми файлами почти всеми мыслимыми способами, позволяя создавать самые разнообразные звуковые программы.

Синхронизация движения текста со звуком

Программа SayPress — это пример программы, в которой синхронизация реализована отображения текста с воспроизведением речи.

Компиляция, компоновка и выполнение программы SayPress

Для компиляции и компоновки программы SayPress выполните следующие действия:

- Убедитесь в том, что ваш ПК находится в защищенном режиме DOS.
- Войдите в директорию c:\spSDK\SampMS\ — для компилятора фирмы Microsoft; или в директорию c:\spSDK\SampBL\ — для компилятора фирмы Borland.
- После приглашения DOS введите:

```
MAKEexe SayPress.c [Enter]
```

Для выполнения программы SayPress:

- Войдите в директорию c:\spSDK\SampMS\ — для компилятора фирмы Microsoft; или в директорию c:\spSDK\SampBL\ — для компилятора фирмы Borland.
- После приглашения DOS введите:

```
SayPress [Enter]
```

В результате работы программы текст "Press any key to continue..." будет отображен синхронно с воспроизведением этой же фразы.

Текст программы SayPress

Текст программы SayPress находится в файле c:\spSDK\SampMS\SayPress.c (или c:\spSDK\SampBL\SayPress.c). Программа воспроизводит звуковой файл пофрагментно. Каждый фрагмент соответствует одному слову из фразы *"Press any key to continue..."*. Для разделения файла на фрагменты можно использовать редактор Sound Editor фирмы TS. Поскольку мы предположили, что у вас нет этого редактора, то приводим разбивку файла c:\spSDK\Sfiles\Press.s:

Байтовая координата	*Слово*
0 - 3896	Press
3896 - 6224	any
6224 - 10137	key
10137 - 11228	to
11228 - SP_END_OF_FILE	continue

Озвучивание текста

Овладев искусством воспроизведения звука, вы сможете создавать очень интересные программы. Например, можно разработать программу, преобразовывающую текст в речь. Для написания такой программы необходимо будет записать все фонемы английского языка. В зависимости от алгоритма, который вы будете использовать, количество фонем может варьироваться (обычно 36 фонем достаточно для конструирования любого английского слова). Например, слово *Bravo* может быть сконструировано из следующих фонем:

```
Bravo = B - R - ah - V - o
```

Файл типа TS (или WAV), используемый такой программой, состоит из 36 фонем. Для воспроизведения определенного слова программа должна разложить его на фонемы, а затем воспроизвести звуковые фрагменты, соответствующие каждой из этих фонем.

Анимация, графика и воспроизведение

Директория c:\spSDK\Util\ содержит две демонстрационные программы — GR1.exe и GR2.exe, которые демонстрируют возможности синхронизации звука с отображением и перемещением графических объектов.

Для выполнения программы GR1.exe выполните следующие действия:

- Войдите в директорию c:\spSDK\Util\.
- После приглашения DOS введите:

```
GR1 путь\имя_файла.s [Enter]
```

Так, для выполнения программы GR1.exe со звуковым файлом Day.s:

- После приглашения DOS введите:

```
GR1 c:\spSDK\Sfiles\Day.s [Enter]
```

Для выполнения программы GR2.exe со звуковым файлом Day.s:

- После приглашения DOS введите:

```
GR2.exe c:\spSDK\Sfiles\Day.s [Enter]
```

Глава 13. Плата Sound Blaster в DOS

В предыдущих главах были представлены различные приложения Windows, использующие звуковую плату типа Sound Blaster. Фактически же эти программы могли использовать любую Windows-совместимую звуковую плату, что является существенным преимуществом оболочки Windows.

Как программисту вам обычно не требуется принимать во внимание то, какая фирма произвела те или иные периферийные устройства вашего компьютера. Если ваша звуковая плата совместима с Windows, то программы будут способны использовать ее. К сожалению, эта ситуация не распространяется на программирование для DOS: здесь звуковые функции определяются производителем звуковой платы. В этой главе мы напишем приложение DOS, называемое PlayVOC.c, — программу, которая воспроизводит звуковые файлы типа VOC с помощью звуковой платы Sound Blaster.

Выполнение программы PlayVOC

Программа PlayVOC использует драйвер Sound Blaster CT-VOICE.DRV, предполагая, что он находится в директории c:\SBpro\DRV\. Перед выполнением PlayVOC вы должны убедиться в том, что драйвер CT-VOICE.DRV располагается именно в этой директории.

Для запуска программы PlayVOC необходимо выполнить следующие действия:

- Войти в c:\spSDK\Samp4SB\.
- После приглашения DOS ввести:

```
PlayVOC  путь\имя_файла.VOC   [Enter]
```

Например, для воспроизведения файла типа .VOC c:\sbpro\vedit2\conga.voc после приглашения DOS введите:

```
PlayVOC c:\sbpro\vedit2\conga.voc   [Enter]
```

Компиляция и компоновка PlayVOC компилятором фирмы Borland

Для компиляции и компоновки PlayVOC.C компилятором Borland C выполните следующие действия:

- Войдите в директорию c:\spSDK\Samp4SB\.
- После приглашения DOS введите:

```
MKBLEXE PlayVOC.c   [Enter]
```

Компиляция и компоновка PlayVOC компилятором фирмы Microsoft

Для компиляции и компоновки PlayVOC.C компилятором Microsoft C необходимо выполнить следующие действия:

- Убедиться в том, что ваш ПК находится в защищенном режиме DOS.
- Войти в директорию c:\spSDK\Samp4SB\.
- После приглашения DOS ввести:

```
MKMSEXE PlayVOC.c   [Enter]
```

Текст программы PlayVOC

Для удобства файл PlayVOC.c располагается в директории c:\spSDK\Samp4SB\. Этот файл совместим как с компилятором фирмы Microsoft, так и с компилятором фирмы Borland.

Загрузка и инициализация драйвера Sound Blaster

Для использования в программе звуковой платы Sound Blaster должен быть загружен драйвер Sound Blaster. Программа PlayVOC использует драйвер Sound Blaster CT-VOICE.DRV, причем предполагает, что он находится в директории c:\SBpro\DRV\.

Драйвер CT-VOICE.DRV загружается в память и инициализируется с помощью функции load_driver().

Функция load_driver() обеспечивает загрузку драйвера. Драйвер c:\SBpro\DRV\CT-VOICE.DRV открывается функцией _dos_open():

```
if ( _dos_open ( C:\\SBpro\\DRV\\CT-VOICE.DRV",
                 O_RDONLY,
                 &handler_to_driver_i) != 0 )
   {
   printf("\n Не могу открыть CT-VOICE.DRV !");
   exit (0);
   }
else
   {
   printf("\n Драйвер был открыт успешно !");
   }
```

Если драйвер CT-VOICE.DRV открывается успешно, то функция load_driver() запрашивает выделение для него памяти:

```
filesize_ul  = filelength ( handler_to_driver_i );
blocksize_ui = (filesize_ul + 15L) / 16;
result_uc = dos_allocmem ( blocksize_ui, &segment_ui );
if ( result_uc !=0 )
   {
   printf ("\n Не могу выделить память для драйвера !");
   return -1;
   }
```

Если выделение памяти происходит успешно, то функция load_driver() загружает драйвер в память:

```
/*--------------------------
 Загружаем драйвер в память.
 --------------------------*/
FP_SEG (pointer_to_driver) = segment_ui;
FP_OFF (pointer_to_driver) = 0;
driver_buffer = (char huge*) pointer_to_driver;
_dos_read ( handler_to_driver_i,
            driver_buffer,
            32000,
            &bytes_read_ui );
_dos_close ( handler_to_driver_i );
```

Обратите внимание на то, что оператор _dos_read() читает максимум 32000 байт. Если объем вашего драйвера превышает 32000 байт, то функция _dos_read() должна вызываться циклически до тех пор, пока драйвер не будет полностью загружен. Однако большинство драйверов имеют объем менее 32000 байт.

После того как драйвер успешно загружен в память, функция load_driver() проверяет, содержат ли байты 3-10 символы CT-VOICE:

```
/*-------------------------------------------------
 Проверяем в загруженном драйвере наличие символов
 "CT-VOICE", начинающихся с четвертого символа.
 -------------------------------------------------*/
temp_s[0] = driver_buffer[3];
temp_s[1] = driver_buffer[4];
temp_s[2] = driver_buffer[5];
temp_s[3] = driver_buffer[6];
temp_s[4] = driver_buffer[7];
temp_s[5] = driver_buffer[8];
temp_s[6] = driver_buffer[9];
temp_s[7] = driver_buffer[10];
temp_s[8] = '\0';
if ( strcmp(temp_s, "CT-VOICE") != 0 )
   {
   printf("\n Не могу найти CT-VOICE ! Совсем плохой драйвер !");
   return -1;
   }
```

Таким образом, мы убеждаемся в том, что загруженный драйвер — это действительно CT-VOICE. Конечно, такая проверка не является полной — ведь проверяются только восемь байтов имени драйвера.

Когда драйвер CT-VOICE.DRV загружен в память, программа может вызывать различные его функции, используя встроенный ассемблер. Для указания конкретной функции драйвера в регистр bx должен быть занесен номер этой функции.

Для инициализации звуковой платы вызывается функция 3:

```
/*----------------------
 Инициализируем драйвер.
 ----------------------*/
__asm {
      mov       bx,3
      call      pointer_to_driver
      mov       out_result,ax
      }
```

Результат инициализации контролируется проверкой значения переменной out_result:

```
/*---------------------------------
 Проверяем результаты инициализации
 ---------------------------------*/
switch ( (int)out_result )
      {
      case 1:   printf ("\n Звуковая плата не обнаружена !");
                return -1;
      case 2:   printf ("\n Неправильный адрес порта !");
                return -1;
      case 3:   printf ("\n Неправильный номер прерывания !");
                return -1;
      }
```

Последнее действие, выполняемое функцией load_driver(), — это установка переменной для связи с платой Sound Blaster. Переменная status_word_i представляет собой целое число, служащее "почтовым ящиком" между программой и звуковой платой. Звуковая плата, изменяя эту переменную, сообщает состояние различных условий. Это означает, что программа PlayVOC должна сообщить звуковой плате адрес этой переменной.

Вначале вычисляется физический адрес переменной status_word_i:

```
/*-------------------------------
Вычисляем адрес слова состояния.
-------------------------------*/
p_status_word_i = &status_word_i;
seg_address = FP_SEG(p_status_word_i);
ofs_address = FP_OFF(p_status_word_i);
```

Затем плата Sound Blaster извещается об этом адресе с помощью вызова функции 5 драйвера CT-VOICE.DRV:

```
/*-------------------------------------
Сообщим драйверу адрес слова состояния.
-------------------------------------*/
__asm {
      mov       bx,5
      mov       es,seg_address
      mov       di,ofs_address
      call      pointer_to_driver
      }
```

Загрузка файла .VOC в память

Файл .VOC загружается в память с помощью функции load_voc().

Функция load_voc() использует функцию _dos_read() для чтения файла .VOC в буфер address_of_voc_buffer[].

Когда буфер заполнится, load_voc(), контролируя два его первых байта, проверяет, является ли загруженный файл файлом .VOC. Все правильные файлы .VOC должны начинаться с символов Creative Voice File. Если загруженный файл — правильный файл .VOC, то первые два символа буфера должны быть Cr:

```
if ( ( address_of_voc_buffer[0] != 'C') ||
     ( address_of_voc_buffer[1] != 'r') )
   {
   printf("\n .VOC должен содержать первыми символами Cr");
   exit(0);
   }
```

Эта проверка не является абсолютной, поскольку проверяются только два символа.

Воспроизведение звукового файла

Файл .VOC воспроизводится с помощью функции play_it().

Функция play_it() вначале разрешает вывод посредством звуковой платы, вызывая функцию номер 4 драйвера CT-VOICE.DRV с регистром al=1:

```
/*-----------------------------
 Разрешим выход звуковой платы.
 -----------------------------*/
__asm {
      mov   bx,4
      mov   al,1
      call pointer_to_driver
      }
```

После этого вычисляются адреса сегмента и смещения буфера файла .VOC:

```
seg_address = FP_SEG ( address_of_voc_buffer );
ofs_address = FP_OFF ( address_of_voc_buffer )
            + length_of_voc_header_uc;
```

И наконец, звуковой плате посылается команда начать воспроизведение из буфера файла .VOC с помощью функции номер 6 драйвера CT_VOICE.DRV:

```
__asm {
      mov  bx, 6                /* Функция номер 6 */
      mov  es, seg_address      /* Адрес сегмента буфера */
      mov  di, ofs_address      /* Адрес смещения буфера */
      call pointer_to_driver    /* Выполнить функцию */
      }
```

Остановка воспроизведения

После запуска воспроизведения функцией play_it() звуковая плата делает все самостоятельно: она продолжает воспроизводить содержимое буфера файла .VOC без какой-либо помощи со стороны программы PlayVOC.

Функция main() программы PlayVOC останавливает воспроизведение, если пользователь нажимает любую клавишу либо если звуковая плата завершает воспроизведение звукового файла.

Напоминаем, что переменная status_word_i в вызове функции load_driver() была указана как почтовый ящик для связи между звуковой платой и программой. Когда воспроизведение завершено, звуковая плата изменяет значение этой переменной на 0.

Состояние звуковой платы проверяется в цикле while(1) функции main(), где анализируется значение переменной status_word_i. Если ее значение — 0, то это означает, что звуковая плата завершила воспроизведение файла .VOC, и в этом случае цикл while(1) завершается.

Воспроизведение завершается и тогда, когда пользователь нажимает любую клавишу на клавиатуре компьютера, причем в этом случае цикл while(1) прерывается и воспроизведение останавливается вызовом функции номер 8 драйвера CT-VOICE.DRV:

```
__asm {
      mov bx,8                  /* Функция номер 8.  */
      call pointer_to_driver /* Выполнить функцию.*/
      }
```

Освобождение буфера файла .VOC и звуковой платы

Перед завершением программы память, выделенная под файл .VOC, освобождается с помощью вызова функции _dos_freemem(), а звуковая плата — с помощью вызова функции release_card():

```
_dos_freemem(FP_SEG(address_of_voc_buffer));
release_card();
```

Функция release_card()

Функция release_card() освобождает звуковую плату, вызывая функцию номер 9 драйвера CT-VOICE.DRV:

```
__asm {
      mov  bx,9                 /* Функция номер 9.  */
      call pointer_to_driver    /* Выполнить функцию.*/
      }
```

Другие функции драйвера Sound Blaster

Как видно из текста программы PlayVOC, знание того, как использовать Sound Blaster, означает знание того, как загрузить драйвер Sound Blaster, как загрузить файл .VOC, как проанализировать слово состояния и как использовать встроенный ассемблер для посылки команд звуковой плате Sound Blaster.

Ниже приведены различные функции драйвера CT-VOICE.DRV, которые вы можете использовать в своих программах.

Функция номер 0: Определение версии драйвера

```
Входы: BX=0 (Номер функции)
Выходы: AH (Номер главной версии)
        AL (Номер субверсии)
```

После выполнения этой функции регистры AH и AL будут содержать номер основной версии и номер субверсии драйвера.

Функция номер 1: Установка адреса порта

```
Входы: BX=1 (Номер функции)
       AX (Адрес порта)
Выходы: Отсутствуют
```

Устанавливает адрес порта платы Sound Blaster. Для того чтобы ее использование было эффективным, она должна вызываться перед функцией 3.

Функция номер 2: Установка номера прерывания

```
Входы: BX=2 (Номер функции)
       AX (Номер прерывания)
Выходы: Отсутствуют
```

Устанавливает номер прерывания платы Sound Blaster. Для того чтобы ее использование было эффективным, она должна вызываться перед функцией 3.

Функция номер 3: Инициализация драйвера

```
Входы: BX=3 (Номер функции)
Выходы: AX=0: Инициализация прошла успешно.
        AX=1: Плата Sound Blaster не обнаружена.
        AX=2: Проблемы с установленным адресом ввода/вывода платы.
        AX=3: Проблемы с установленным номером прерывания.
```

Функция номер 4: Включение и выключение внешнего динамика

```
Входы: BX=4 (Номер функции)
       AL=0 (Динамик выключен)
       AL=1 (Динамик включен)
Выходы: Отсутствуют
```

Функция номер 5: Установка адреса слова состояния

```
Входы: BX=5 (Номер функции)
       ES:DI (Адрес слова состояния)
Выходы: Отсутствуют
```

Функция номер 6: Начать воспроизведение

```
Входы:  BX=6 (Номер функции)
        ES:DI (Адрес буфера .VOC)
Выходы: Отсутствуют
```

Функция номер 7: Начать запись

```
Входы:  BX=7 (Номер функции)
        AX   (Частота дискретизации)
        DX:CX (Максимальная длина записи)
        ES:DI (Адрес буфера .VOC)
Выходы: Отсутствуют
```

Функция номер 8: Остановить воспроизведение или запись

```
Входы:  BX=8 (Номер функции)
Выходы: Отсутствуют
```

После выполнения этой функции слово состояния будет равно 0.

Функция номер 9: Отключить драйвер

```
Входы:  BX=9 (Номер функции)
Выходы: Отсутствуют
```

Функция номер 10: Пауза в воспроизведении

```
Входы:  BX=10 (Номер функции)
Выходы: AX=0 (Пауза успешно выполнена)
        AX=0 (Пауза не выполнена)
```

После выполнения этой функции воспроизведение прерывается.

Функция номер 11: Возобновление воспроизведения

```
Входы:  BX=10 (Номер функции)
Выходы: AX=0 (Возобновление успешно выполнено)
        AX=0 (Возобновление не выполнено)
```

После выполнения этой функции воспроизведение, которое было остановлено функцией номер 10, возобновляется.

Создание программ мультимедиа для DOS с помощью звуковой платы Sound Blaster

Для создания программ мультимедиа для DOS с помощью Sound Blaster можно использовать ту же технику, которая была применена в приложениях Windows. Таким образом, необходимо разработать таблицу, в которой определяется, какой звуковой фрагмент соответствует каждому фрейму. Во время воспроизведения отображается кадр, соответствующий звуковому фрагменту, воспроизводимому в данный момент.

Приложение. Звуковые динамически подключаемые библиотеки

В этой книге рассмотрены вопросы создания приложений Windows, которые используют статическую библиотеку, размещенную в директории c:\spSDK\TegoWlib. Альтернативным способом является создание приложений Windows, которые используют динамически подключаемые библиотеки (DLL).

У программ, применяющих библиотеки DLL, есть как достоинства, так и недостатки.

Достоинства применения звуковых функций из библиотек DLL

Предположим, что ваше приложение состоит из нескольких программ, каждая из которых вызывает sp_-функции. В этом случае лучше использовать библиотеку DLL.

Библиотечный файл .DLL расположен на винчестере и используется всеми приложениями, которые вызывают находящиеся в нем функции. Поскольку эти функции не размещены непосредственно в вашем приложении, то размер приложения уменьшается.

Использование звуковых библиотек DLL из Visul Basic и других языков программирования под Windows

Еще одним преимуществом применения библиотек DLL является то, что они могут быть использованы и программами, написанными на других языках программирования для Windows, например Visual Basic для Windows.

Недостатки применения звуковых библиотек DLL

К недостаткам применения звуковых библиотек DLL относится то, что для нормальной работы вашего приложения потребуется, чтобы соответствующая DLL была установлена на винчестере конечного пользователя. Следовательно, необходимо, чтобы дистрибутивная дискета вашего приложения содержала эту библиотеку, а программа инсталляции вашего приложения копировала ее с дискеты на винчестер конечного пользователя.

Применение звуковых DLL

Перепишем приложение HearMe так, чтобы оно использовало вместо статической библиотеки DLL звуковую библиотеку, которая называется TegoSND.DLL. На прилагаемой к книге дискете этой библиотеки нет — ее можно получить только в фирме TegoSoft.

Применение динамически присоединяемой библиотеки DLL4Snd.DLL

Преобразовать приложение, использующее статическую библиотеку TegoWin.lib, в приложение, которое использует TegoSnd.DLL, достаточно просто.

Ниже описано, как осуществить это преобразование в отношении программы HearMe.

Шаг 1

Для применения DLL необходимо использовать файл TegoSND.h. Так, в файле HearMe.c надо заменить оператор

```
#include "c:\spSDK\TegoWlib\sp4Win.h"
```

оператором

```
#include "c:\spSDK\DLL\TegoSND.h"
```

Примечание:

Файла TegoSND.h на прилагаемой дискете нет.

В файле TegoSND.h находятся прототипы объявлений sp_-функций и часть DLL.

Шаг 2

Приложение использует sp_-функции из DLL. Следовательно, в DEF-файле должно находиться соответствующее объявление IMPORT, в котором описываются все sp_-функции, которые импортируются приложением из DLL.

Поскольку программа HearMe использует только две sp_-функции, то приведенное ниже объявление IMPORT содержит все, что нужно импортировать из DLL:

```
TegoSND.sp_OpenSession
TegoSND.sp_PlayF
```

Шаг 3

Файл HearMe.mak тоже нужно изменить, так как после подключения DLL в применении статической библиотеки c:\spSDK\TegoWlib\TegoWin.lib больше нет необходимости.

Ниже приведен измененный файл HearMe.mak:

```
#============
# Hearme.mak
#============
Hearme.exe : Hearme.obj Hearme.h Hearme.def Hearme.res
    link /nod Hearme.obj, Hearme.exe, NUL, \
        slibcew.lib oldnames.lib libw.lib commdlg, \
        Hearme.def
   rc -t Hearme.res

Hearme.obj : Hearme.c Hearme.h
   cl -c -G2sw -Ow -W3 -Zp  Hearme.c
Hearme.res : Hearme.rc Hearme.h Hearme.ico
   rc -r Hearme.rc
```

Вот и все изменения, которые нужно внести в приложение HearMe, для того чтобы оно использовало библиотеку DLL.

О

П

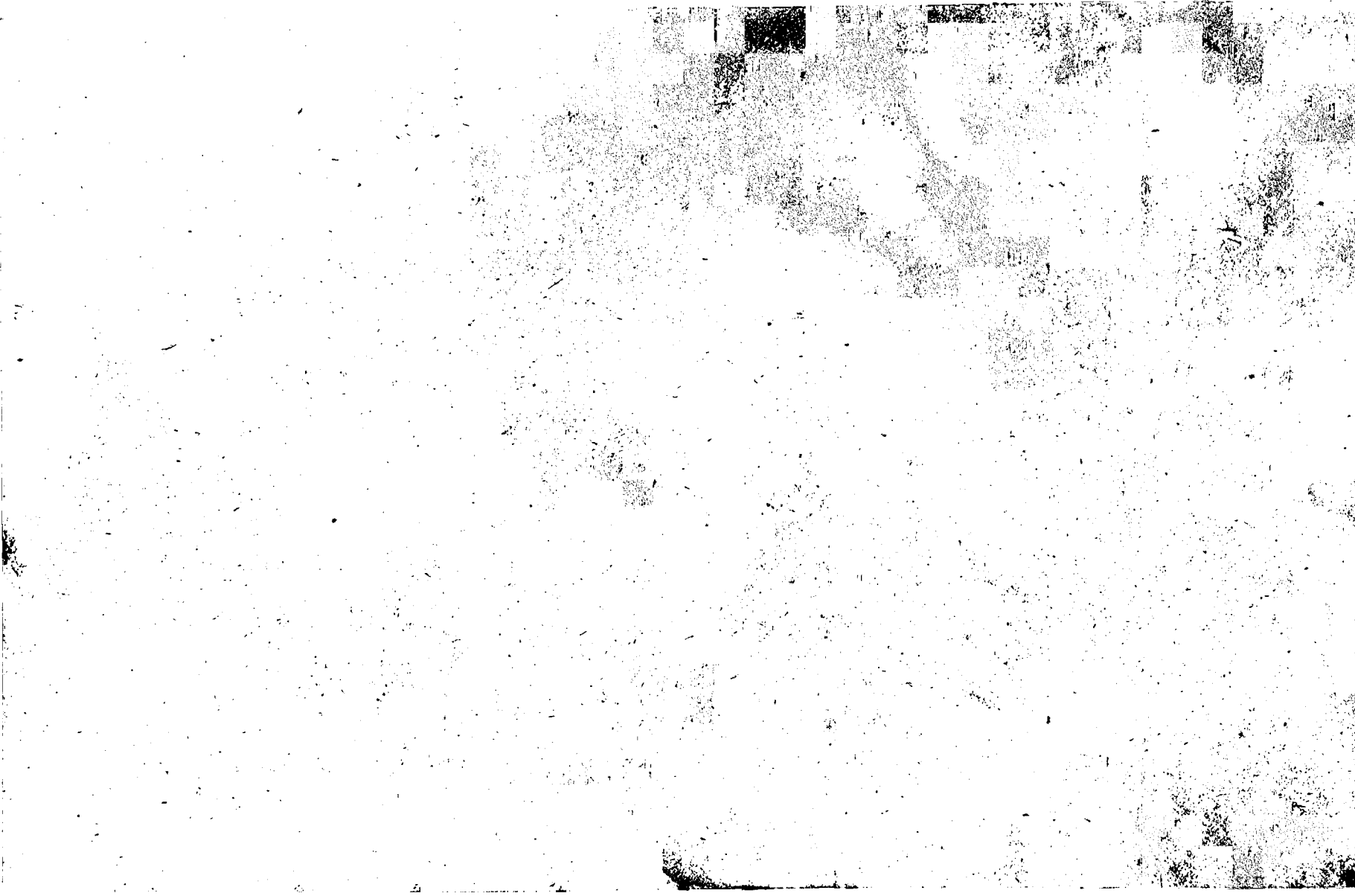
Р

С

Ф

У

Ц



Содержание

ИЗДАТЕЛЬСТВО

БИНОМ

103473 МОСКВА-473 а/я 133.
Факс (095) 211-1114
Телефоны
(095) 211-1690
(095) 211-4344

Приглашаем к сотрудничеству авторов и научных редакторов в области электроники и программирования

Принимаются заказы на размещение рекламы в наших изданиях

Натан Гуревич, Ори Гуревич

ПРОГАММИРОВАНИЕ ЗВУКА ДЛЯ DOS И WINDOWS

Подписано в печать 22.08.95. Формат 70x108 1/16. Печ. л. 25.
Бумага офсетная. Печать офсетная.
Тираж 12000 экз. Заказ 1052

Издательство «БИНОМ», 1995 г.
Москва, ул. Новослободская, д. 50/1, стр. 1а.
Лицензия на издательскую деятельность № 063367 от 20 мая 1994 г.

Издательство «НАУЧНАЯ КНИГА», 1995 г.
Москва, ул. Автозаводская, д. 16.
Лицензия на издательскую деятельность № 071152 от 14 апреля 1995 г.

Отпечатано с готовых диапозитивов в полиграфической фирме
«Красный пролетарий»
103473, Москва, Краснопролетарская, 16