Лабораторная работа №7

Использование ресурсов и DLL

Работа выполняется в системе программирования Borland Delphi 2.0 в среде Windows 95 с использованием средств пакета Borland Resource Workshop, а также утилиты Image Editor и компилятора ресурсов из состава Delphi. Использование средств визуального программирования и классов VCL не допускается, то есть Delphi используется просто как 32-разрядный компилятор с языка Паскаль для операционной системы Windows.

Теоретические положения

Форматы исполнимых файлов Windows (.EXE и .DLL) подразумевают возможность хранения в теле этих файлов различных данных. Чаще всего это небольшие битовые образы, используемые в качестве пиктограмм, значков и курсоров мыши, а также текстовые строки. Эти данные располагаются не в области данных программы, не загружаются автоматически в память при загрузке модуля и недоступны через идентификаторы переменных внутри программы. Для доступа к этим данным их необходимо специальным образом загрузить из тела исполнимого файла, как если бы они располагались в дисковых файлах.

Для описания таких данных используются файлы ресурсов. Файлы ресурсов обычно имеют расширение .RES (в Delphi – еще и .DCR) и присоединяются к телу исполнимого файла на этапе компоновки, одновременно с файлами .OBJ, а также .LIB (для языка C) или .DCU (для Delphi).

Файлы ресурсов — это бинарные файлы, имеющие определенный формат; для создания и редактирования ресурсов используется компилятор ресурсов (например в Delphi — brc32.exe), позволяющий на основе текстового описания ресурсов (обычно это фалы с расширением .RC) построить бинарный файл .RES. Кроме того, существуют утилиты, позволяющие редактировать стандартные типы ресурсов непосредственно в файле .RES, а также осуществлять декомпиляцию ресурсов в текстовое описание. В этой работе для редактирования ресурсов предлагается использовать пакет Borland Resource Workshop.

Стандартными типами ресурсов являются:

- битовые образы (bitmaps)
- значки (icons)
- курсоры мыши (cursors)
- символьные строки (strings)
- меню (menus)
- комбинации "горячих клавиш" (keyboard accelerators)
- диалоги (dialog boxes)

Кроме того, можно создать ресурсы произвольного формата (ресурсы пользователя).

Подключение файлов ресурсов к программе

Для включения ресурсов в программу на Delphi файл ресурсов должен быть в явном виде упомянут в программе с использованием директивы компилятора $RESOURCE ($R), например так:

```
{$R myres.res}
```

К одной программе может быть подключено любое количество файлов ресурсов. При работе с Delphi не следует называть файл ресурсов именем проекта (главного файла), так как

такой ресурс Delphi создает автоматически, и созданный программистом ресурс будет просто стерт.

В модулях (units) языка Паскаль при компиляции модулей имена необходимых файлов ресурсов просто заносятся в тело файлов .DCU; поэтому при компоновке файлов .EXE или .DLL все файлы .RES, используемые в модулях, должны быть доступны компилятору. Ресурсы включаются в исполнимый файл только на этапе его компоновки, они не присутствуют в модулях .DCU.

Компиляция ресурсов

Компиляция текстового описания ресурсов в файл .RES осуществляется из командной строки компилятором ресурсов BRC32 (Borland Resource Compiler). Вызов (маршрут к директории расположения BRC32 должен быть прописан в строке PATH или указан в команде)

brc32 -r myres.rc

приведет к компиляции описания ресурсов myres.rc и созданию файла ресурсов myres.res.

Параметр **-r** означает, что не требуется производить немедленное включение файла ресурсов myres.res в исполнимый модуль. С опциями компилятора ресурсов можно ознакомиться, запустив brc32 без параметров.

Утилиты для работы с ресурсами

Утилита Image Editor, входящая в комплект поставки Borland Delphi, позволяет создавать и манипулировать простейшими бинарными ресурсами: битовыми образами, значками и курсорами мыши. Кроме того, она позволяет создавать и редактировать битовые образы, значки и курсоры непосредственно в теле файла с расширением .RES.

Утилита Resource Workshop кроме того позволяет работать как с бинарными (.RES), так и с текстовыми (.RC) файлами ресурсов, а также с ресурсами в теле исполнимого модуля, осуществлять их компиляцию/декомпиляцию, а также работать с ресурсами-меню, диалогами и таблицами строк.

Справка по Resource Workshop содержит также описание синтаксиса языка описания ресурсов, применяемого в файлах .RC.

Описание и использование ресурсов

Курсоры, значки и битовые образы

Битовый образ — это изображение в формате BMP. Можно использовать изображения любого размера и глубины цвета. Обычно изображения создаются в графическом редакторе и сохраняются в формате .BMP.

Значок (icon) — это растровое изображение размером 32x32 или 16x16 пикселов, монохромное или 16-цветное (в современных версиях Windows поддерживается и большая глубина цвета). В дополнение к возможным 2 или 16 цветам, пикселы значка могут быть прозрачными и инвертирующими. Прозрачные пикселы сохраняют цвет пикселов изображения, поверх которого выводится значок, инвертирующие — инвертируют. Значки хранятся в файлах с расширением .ICO. Гораздо меньшее количество графических редакторов поддерживают этот формат, в частности его поддерживают Image Editor в Delphi и Resource Workshop.

Курсор мыши — это монохромное (в современных версиях Windows допускается цветное) растровое изображение размером 32x32 пиксела. Каждый пиксел курсора может быть черным, белым, прозрачным или инвертирующим. Так как курсор предназначен для отображения текущего положения указателя мыши, в нем обязательно задается так называемая вершина (hotspot) в виде пары координат относительно левого верхнего угла битового образа курсора. При отображении положения мыши вершина курсора совмещается с текущими координатами мыши. Обычно курсоры в виде стрелки имеют вершину на конце стрелки, курсоры в виде перекрестия — в центре перекрестия и т.д. Курсоры хранятся в виде файлов с расширением .CUR. Обычно для их создания требуются специализированные графические редакторы (например Image Editor в Delphi или Resource Workshop).

При использовании компилятора ресурсов каждый значок, битовый образ или курсор обычно описываются в виде ссылки на соответствующий файл ресурса в формате

<идентификатор> <ТИП> <файл>

например:

```
picture1    BITMAP    pic1.bmp
hand        CURSOR    hand.cur
appicon     ICON      icon1.ico
```

Здесь picture1, hand и appicon — идентификаторы битового образа, курсора мыши и значка соответственно. Вместо строковых идентификаторов при описании ресурсов могут использоваться целые положительные 16-разрядные числа. Кроме того, описание битового образа ресурса может располагаться не в отдельном файле, а в файле .RC в виде шестнадцатеричного дампа памяти (такая ситуация возникает обычно в результате декомпиляции файлов .RES).

Для использования ресурсов в программе их необходимо загрузить.

Битовые образы загружаются с помощью функции

function LoadBitmap(hInstance: HINST; lpBitmapName: PAnsiChar): HBITMAP;

Функция возвращает хэндл битового образа или 0 при неудаче. В качестве первого параметра указывается хэндл экземпляра приложения, из исполнимого файла которого производится загрузка ресурса. Указав 0 в качестве hInstance можно получить доступ к некоторым стандартным битовым образам Windows (стрелки полос прокрутки и т.п., см. Справку по Win32 API). Второй параметр должен содержать либо указатель на строковой идентификатор битового образа, либо его числовой идентификатор, если в файле ресурса образ описывалась с номером. Windows определяет, передан указатель на строку или числовой идентификатор, проверяя старшее слово переданного указателя: если передано 16-битное число, то старшее слово нулевое, в то время как указатель на область данных программы всегда имеет ненулевое старшее слово.

После использования каждый загруженный с помощью LoadBitmap битовый образ должен быть уничтожен с помощью функции DeleteObject.

Для загрузки значков используется функция

function LoadIcon(hInstance: HINST; lpIconName: PChar): HICON;

Ее использование аналогично функции LoadBitmap. В отличие от функции LoadBitmap, если указанный значок уже был ранее загружен, вторая его копия не будет создаваться и будет возвращен хэндл уже загруженного значка. Загруженные с помощью LoadIcon значки не нуждаются в уничтожении.

Загруженный значок может использоваться при описании классов окон. Также имеется возможность отобразить значок в контексте устройства с помощью функции

DrawIcon(hdc,x,y,hIcon);

Курсоры загружаются при помощи функции

function LoadCursor(hInstance: HINST; lpCursorName: PAnsiChar): HCURSOR;

Она аналогична функции LoadIcon. Хэндл курсора может использоваться при описании классов окон. Кроме того, при обработке сообщения WM_MOUSEMOVE имеется возможность анализировать положение мыши и менять форму курсора при помощи функции

SetCursor(hCursor);

Если эту функцию не вызывать, то будет использован курсор, определенный в оконном классе (точнее – при любом движении мыши в окне Windows устанавливает курсор в соответствии с описанием оконного класса, поэтому если предполагается "вручную" менять форму курсора в оконной процедуре, то в классе окна в качестве хэндла курсора лучше указывать 0).

Пусть имеется файл описания ресурсов, содержащий описание:

```
123    BITMAP    pic1.bmp
ic1    ICON      icon1.ico
```

Тогда соответствующие битовый образ и значок должны быть загружены при помощи вызовов:

```
hbmp := LoadBitmap(hInstance, pointer(123) );
hicon := LoadIcon(hInstance, 'ic1');
..................... операции ..............
DeleteObject(hbmp);
```

Идентификаторы должны быть уникальными для однотипных ресурсов. Это означает, что среди всех файлов ресурсов программы не должно быть, например, двух битовых образов с идентификатором 'pic1', хотя наличие битового образа и значка с одинаковым идентификатором допускается. Тем не менее, рекомендуется всегда использовать уникальные идентификаторы ресурсов.

Таблицы строк символов

Таблицы строк символов являются также распространенным типом ресурсов. Использование этого вида ресурсов позволяет существенно облегчить процесс локализации программного обеспечения (например, перевод с английского на русский язык). Если все приглашения и сообщения, которые программа выводит на экран, хранятся в файлах ресурсов, то для переводчика исчезает необходимость работать с исходным текстом программы — достаточно лишь перевести собранные в одном месте строки. Однако отлаживать программу, все текстовые строки которой находятся в отдельном файле ресурсов, не очень удобно.

Символьные строки организуются в таблицы и объявляются с использованием ключевого слова STRINGTABLE следующим образом:

```
STRINGTABLE
{
<id1>, "<string1>"
.....................
}
```

Каждая строка в таблице записывается на отдельной строке и заключается в двойные кавычки (как в языке C), перед каждой строкой, отделенное запятой, указывается число-идентификатор. Например, может быть объявлена следующая таблица строк:

```
STRINGTABLE
{
1, "Ошибка времени выполнения"
2, "Недопустимая операция"
10, "Нормальное завершение"
}
```

Использование литеральных констант неудобно: необходимо поддерживать соответствие номеров строк в программе и в файле ресурсов. В языке C эта проблема решается за счет создания файла заголовка с расширением .H, в котором определяются числовые константы, и который подключается к файлу .RC и исходному тексту программы директивой #include. При работе с Delphi применяется такой же подход: константы, идентифицирующие символьные строки, могут быть описаны в отдельном файле в соответствии с синтаксисом языка Паскаль в виде секции **const,** и затем включены в исходный текст программы с помощью директивы компилятора $INCLUDE ($I), а в файл .RC — с помощью #include. Можно также объявить модуль (unit) языка Паскаль, интерфейсная часть которого состоит из объявлений констант, и подключить его к программе в разделе **uses**, а к описанию ресурсов — с помощью #include.

Тогда могут быть созданы файлы (Resource Workshop "умеет" делать это автоматически при работе со строками, но он, к сожалению, не всегда корректно работает с русским алфавитом):

```
——— константы ресурсов: RESCONST.PAS ———
unit resconst;
interface
const
  IDS_Runerror = 1;
  IDS_InvalidOp = 2;
  IDS_Normalterm = 10;
implementation end.
———— описание ресурсов ————————
#include "resconst.pas"
STRINGTABLE
{
IDS_RUNERROR, "Ошибка времени выполнения"
IDS_INVALIDOP, "Недопустимая операция"
IDS_NORMALTERM, "Нормальное завершение"
}
————————————————————————————
```

Строки загружаются с помощью функции LoadString в готовый буфер:

function LoadString(hInst: HINST; uID: UINT; lpBuffer: PChar; nBufferMax: Integer): Integer;

Здесь hInst — хэндл экземпляра приложения, из исполнимого файла которого загружается строка, uID — целочисленный идентификатор строки, lpBuffer — указатель на буфер для приема данных, nBufferMax — размер буфера. Если загружаемая строка превышает размер буфера, она будет усечена. Функция возвращает количество прочитанных символов. Загружаемая строка, даже усеченная, всегда заканчивается нулевым символом.

Ресурсы пользователя

Имеется возможность сохранить в виде ресурса и подключить к исполняемому модулю произвольные данные. Для этого в файле описания ресурсов должна быть ссылка на файл, содержащий подключаемые данные, например:

```
data1       MYDATA    example1.dat
```

В качестве идентификатора и типа данных может быть указана произвольная строка. Загрузить такие данные можно при помощи вызова

hData := LoadResource (hInstance, FindResource(hInstance, 'MYDATA', 'data1'));

Для работы с самими данными необходим не хэндл, а указатель, что выглядит примерно так:

pData := LockResource(hData);
{действия с pData}

Вызов функции UnlockResource, использовавшейся в Windows 3.x, в Windows 95 не требуется. Когда ресурс больше не нужен, его следует освободить при помощи

FreeResource(hData);

Подробнее о функциях LoadResource, FindResource, LockResource и FreeResource см. в Справке по Win32 API.

Меню

Простейшим способом включения меню в интерфейс Windows-программы является описание его в файле ресурсов и последующая его загрузка оттуда. Существует также множество функций API, позволяющих создавать и изменять меню в процессе работы программы, однако большинство из них здесь рассматриваться не будут.

Меню, как и дочерние окна управления, при различных действиях пользователя посылает программе сообщения WM_COMMAND.

Меню бывают двух видов: главное меню (main menu), отображаемое в виде строки или нескольких строк вверху окна, подменю (drop-down menu), выглядящие как вертикальные списки строк и появляющиеся при выборе пунктов в главном меню, а также всплывающие меню (popup menu) появляющиеся (новый стандарт пользовательского интерфейса) при щелчке правой кнопкой мыши по тому или иному органу управления в окне.

Пункты меню могут быть "разрешены" (enabled), "запрещены" (disabled) или "недоступны" (grayed — серые). При выборе пользователем разрешенных пунктов меню происходят какие-то действия (открывается всплывающее меню или программе посылается сообщение WM_COMMAND). Запрещенные и недоступные пункты меню также можно выбрать, но никаких действий при этом не происходит. Запрещенные пункты меню, в отличие от недоступных, не отображаются серым цветом и выглядят так же, как разрешенные.

Кроме того, пункты всплывающего меню могут быть "помечены" (checked), при этом слева от соответствующего пункта меню отображается значок "галочки".

Описание меню в файле ресурсов производится согласно следующему шаблону:

```
<имя_меню> MENU
{
[<список элементов меню>]
}
```

<имя_меню> — это строка или число, служащее для идентификации меню в программе при описании класса окна. Список элементов меню состоит из описаний пунктов меню или всплывающих меню.

Пункт меню (которому не соответствует подменю) описывается как

```
MENUITEM "<текст>", <идентификатор> [, <признаки>]
```

Если пункту меню соответствует подменю, то оно описывается как

```
POPUP "<текст>" [, <признаки>]
{
[<список элементов меню>]
}
```

<Текст>, заключенный в кавычки — это строка, которая отображается в описываемом пункте меню. В текст может быть включен знак "&"; символ текста, следующий за амперсантом, будет отображаться подчеркнутым и при нажатии его на клавиатуре совместно с клавишей Alt будет выбираться этот пункт меню.

<идентификатор> — это число, передаваемое в программу в параметре сообщения WM_COMMAND при выборе этого пункта меню. Удобно использовать внешние определения идентификаторов, как это было описано выше при обсуждении строковых таблиц, и подключать их к описанию ресурсов директивой #include. Имена констант для идентификаторов меню принято начинать с символов "IDM_", например IDM_EDIT.

В качестве признаков элементов главного меню используются следующие флаги, которые можно объединять с помощью символа дизъюнкции ("ИЛИ") языка C ("|" — "трубопровод"):

GRAYED - пункт недоступен и выводится серым.
INACTIVE - пункт неактивен (не генерирует WM_COMMAND), но отображается обычным образом, несовместимо с GRAYED.
MENUBREAK - следующие пункты меню отображаются с новой строки.
HELP - этот и все следующие пункты меню "прижимаются" к правому краю строки меню.

В описании всплывающего меню могут также встречаться как подменю, так и пункты меню, описываемые точно так же с помощью директив POPUP и MENUITEM. Кроме того, внутри всплывающих меню можно нарисовать горизонтальную черту, включив в качестве описания строки меню строку

```
MENUITEM SEPARATOR
```

Может дополнительно использоваться признак CHECKED, означающий, что пункт меню выбран и отмечен галочкой.

Использование меню

Для указания на использование меню из ресурса в описании оконного класса достаточно указать его имя, как оно описано в ресурсе, например

```
wndclass.lpszMenuName := 'MyMenu';
```

или, если при описании меню использовался числовой идентификатор, например, 12:

```
wndclass.lpszMenuName := pointer(12);
```

или

wndclass.lpszMenuName := '#12';

Все окна этого класса будут иметь указанное меню. Чтобы указать свое меню для конкретного окна при его создании, необходимо загрузить описание меню из ресурса:

hMenu := LoadMenu(hInstance, 'MyMenu');

и затем передать полученный хэндл меню hMenu в качестве девятого параметра функции CreateWindow.

Наконец, имея хэндл загруженного описания меню, можно изменить меню уже существующего окна, вызвав функцию

SetMenu(hwnd, hMenu);

а просто узнать хэндл меню окна — с помощью функции

GetMenu(hWnd);

Любое связанное с окном меню уничтожается при уничтожении окна. Однако загруженные, но не связанные с окнами меню должны быть явно удалены при помощи DestroyMenu(hMenu).

Основные сообщения меню

При выборе разрешенного пункта меню (щелчке по пункту меню левой кнопкой мыши или нажатии клавиши Enter) программе посылается сообщение WM_COMMAND с параметрами

loWord(wParam) - числовой идентификатор пункта меню
hiWord(wParam) = 0; lParam = 0

При выборе пункта системного меню (открываемого при щелчке по значку в левом верхнем углу окна) посылается сообщение WM_SYSCOMMAND с теми же параметрами. Для анализа команды необходимо замаскировать младшие 4 бита младшего слова wParam, т.е. выражение (loword(wParam) and $FFF0) может принимать значения sc_size, sc_move, sc_minimize, sc_maximize, sc_close и др. Сообщение WM_SYSMENU обычно передается в DefWindowProc.

При каждом перемещении указателя по меню при помощи мыши или клавиш управления курсором программе посылается сообщение WM_MENUSELECT. Оно полезно, например, для вывода подсказки по выбранному пункту меню в строке состояния окна. Параметры:

loword(wParam) - числовой идентификатор пункта меню или хэндл всплывающего меню
hiword(wParam) - флаги выбора
lParam - хэндл меню, содержащего выбранный пункт

Флаги выбора содержат логическую сумму констант MF_GRAYED, MF_DISABLED, MF_CHECKED, MF_POPUP, MF_SYSMENU, MF_MOUSESELECT и др.

Перед автоматическим выводом на экран всплывающего меню программа получает сообщение WM_INITMENUPOPUP, позволяющее привести содержимое меню в надлежащий вид, например — разрешить или запретить некоторые его пункты. Параметры:

wParam - хэндл всплывающего меню
loword(lParam) - индекс всплывающего меню
hiword(lParam) - 1 для системного меню и 0 для обычного

Для простейшего управления меню используются вызовы

```
EnableMenuItem(hMenu, id, MF_GARAYED);
EnableMenuItem(hMenu, id, MF_ENABLED);
CheckMenuItem(hMenu, id, MF_UNCHECKED);
CheckMenuItem(hMenu, id, MF_CHECKED);
```

Для управления меню (в том числе системным меню окна) и создания меню также используются следующие функции, с описанием которых можно ознакомиться в Справке по Win32 API:

CreateMenu, CreatePopupMenu, AppendMenu, GetSystemMenu, DeleteMenu, RemoveMenu, InsertMenu, ModifyMenu, DrawMenuBar, GetSubMenu, GetMenuItemCount, GetMenuItemID, TrackPopupMenu.

Таблица акселераторов

Таблицы акселераторов используются для упрощения построения интерфейса клавиатуры. В этой таблице задаются комбинации клавиш (как правило, соответствующие пунктам меню, хотя это необязательно), нажатие которых автоматически приводит к генерации сообщений WM_COMMAND с тем или иным кодом.

В файле описания ресурсов таблицы акселераторов определяются следующим образом:

```
<имя_таблицы> ACCELERATORS
{
[определения комбинаций клавиш]
}
```

В качестве имени таблицы можно указывать символьный или числовой идентификатор, как и для меню. Определения клавиш — это строки одного из четырех возможных форматов:

```
"<символ>", <числовой_идентификатор> [, SHIFT] [, CONTROL] [, ALT]
"^<символ>", <числовой_идентификатор> [, SHIFT] [, CONTROL] [, ALT]
<ASCII-код>,  <числовой_идентификатор>, ASCII [, SHIFT] [, CONTROL] [, ALT]
<виртуальный_код>, <числовой_идентификатор>, VIRTKEY [, SHIFT] [, CONTROL] [,
ALT]
```

Числовой идентификатор быстрой клавиши передается в параметрах сообщения WM_COMMAND. Слова SHIFT, CONTROL и ALT означают, что указанные регистровые клавиши образуют комбинацию с клавишей, код которой указывается в первом параметре.

Первый способ определения клавиши подразумевает, что в первом параметре указан символ, соответствующий алфавитно-цифровой клавише, второй — что это комбинация символа с клавишей Ctrl (фраза CONTROL в этом случае ничего не меняет), в третьем случае первым параметром должен быть числовой код буквы, в последнем случае — виртуальный код клавиши Windows. Наиболее употребительными являются первый и четвертый варианты.

Использование таблицы акселераторов

Для использования описанных в файле ресурсов комбинаций клавиш их необходимо загрузить, а также несколько модифицировать цикл выборки сообщений.

Загрузка таблицы акселераторов и получение ее хэндла осуществляется с помощью функции LoadAccelerators:

hAccel := LoadAccelerators(hInstance, 'MyAccel');

Цикл обработки сообщений дополняется вызовом функции проверки комбинаций клавиш по таблице:

```
while getMessage(msg, 0,0,0) do
  if not TranslateAccelerator(hwnd, hAccel, msg) then begin
    TranslateMessage(msg);
    DispatchMessage(msg);
  end;
```

Функция TranslateAccelerator проверяет, является ли сообщение msg сообщением клавиатуры и, если это так, проводит поиск комбинации нажатых клавиш в таблице акселераторов. Если комбинация найдена, вызывается оконная процедура окна hwnd и ей передается сообщение WM_COMMAND с кодом, идентифицирующим нажатую комбинацию, функция при этом возвращает TRUE. Для сообщений, не приводящих к посылке сообщения WM_COMMAND при трансляции, функция возвращает FALSE.

Параметр hwnd функции TranslateAccelerator позволяет сосредоточить обработку акселераторов в одной оконной процедуре: какое бы окно не являлось получателем сообщения msg (хэндл окна получателя содержится в теле сообщения), сообщение WM_COMMAND будет передано в оконную процедуру принудительно указанного окна hwnd. В качестве hwnd программа обычно должна передавать либо хэндл главного окна, либо хэндл ее активного окна.

Сообщение WM_COMMAND, порождаемое при нажатии быстрой клавиши, имеет следующие параметры:

loword(wParam) - числовой идентификатор комбинации клавиш из таблицы акселераторов

hiword(wparam) = 1; lParam = 0;

Если акселератор соответствует тому или иному пункту меню (имеет тот же идентификатор), то помимо сообщения WM_COMMAND той же оконной процедуре посылаются сообщения активизации меню, т.е. WM_INITMENU, WM_INITMENUPOPUP, WM_MENUSELECT. Если клавиша соответствует запрещенному пункту меню, то сообщения WM_COMMAND и прочие не посылаются.

Окна диалога

В файле ресурса могут быть также описаны диалоговые окна программы. Диалоговые окна как правило предназначены для ввода пользователем данных, которые невозможно или неудобно вводить при помощи меню. Диалоговые окна содержат дочерние окна управления (кнопки, поля ввода текста, радио-кнопки и кнопки-флажки), в описании вида и положения которых и состоит описание диалога.

При описании диалога требуется задавать множество координат, и это неудобно делать вручную, а еще неудобнее редактировать такое описание. Кроме того, координаты в диалогах, описываемых в ресурсах, задаются не в физических пикселах, а в особой координатной сетке (с шагом 1/8 высоты шрифта диалога по вертикали и 1/4 ширины шрифта по горизонтали), из-за чего диалоги при разных разрешениях экрана выглядят примерно одинаково по размерам. Для облегчения процесса описания диалогов (как, впрочем, и меню) созданы различные инструменты, в частности такая функция есть в пакете Borland Resource Work-

shop. С его помощью можно "нарисовать" внешний вид окна диалога и затем сохранить в виде текстового описания ресурса или непосредственно в бинарном файле ресурса.

Для обеспечения функционирования диалога необходимо описать так называемую процедуру диалога, которая очень похожа на оконную процедуру. Принципиальное отличие между диалоговой процедурой и обычной оконной процедурой состоит в том, что ее вызов производится из настоящей оконной процедуры окна диалога, которая находится "в недрах" Windows; и диалоговой процедуре передаются не все сообщения, которые получает оконная процедура. Заголовок диалоговой процедуры должен быть следующим (имя процедуры — любое):

```
function DlgProc(hDlg: HWND; Msg: UINT;
                 wParam: WPARAM; lParam: LPARAM): longint; stdcall;
```

Если функция обрабатывает сообщение, она должна возвращать ненулевое значение, иначе — 0. Сообщения, которые не обрабатываются, не должны передаваться ни в DefWindowProc, ни в DefDlgProc, так как их обработка и так сосредоточена в оконной процедуре диалога. Вызов DefDlgProc внутри диалоговой процедуры приводит к рекурсии (оттуда снова будет вызвана диалоговая процедура), в результате чего приложение намертво "подвешивает" операционную систему.

Процедура диалога обычно обрабатывает как минимум два сообщения: WM_INITDIALOG, которое посылается диалоговому окну при его инициализации, и WM_COMMAND — сообщения от дочерних окон управления.

Если в ответ на WM_INITDIALOG процедура диалога возвращает TRUE, то Windows дает фокус первому из элементов управления диалога (его хэндл передается в wParam); при инициализации диалога можно изначально дать фокус другому элементу управления с помощью функции SetFocus, при этом обработчик WM_INITDIALOG должен вернуть FALSE.

При попытке использовать SetFocus возникает проблема: так как диалоговое окно создается Windows, то в программе неизвестны хэндлы дочерних органов управления. Однако эту информацию можно легко получить, зная числовой идентификатор органа управления, задаваемый в файле ресурса, при помощи функции GetDlgItem:

```
function GetDlgItem(hDlg: THandle; id: integer): THandle;
```

Сообщения WM_COMMAND от дочерних окон управления и способы управления ими достаточно подробно рассмотрены в теоретическом материале к предыдущей работе.

Для посылки сообщений органам управления в API реализована функция

```
function SendDlgItemMessage(hDlg: HWND; nIDDlgItem: Integer;
                 Msg: UINT; wParam: WPARAM; lParam: LPARAM): Longint;
```

работающая аналогично SendMessage, при этом получателем сообщения является дочернее окно диалога hDlg с идентификатором nIDDlgItem (при этом не требуется знать хэндл этого органа управления).

В окнах диалога Windows автоматически поддерживает интерфейс клавиатуры, позволяющий перемещать фокус от одного дочернего окна управления к другому при помощи клавиши Tab и клавиш управления курсором.

Диалог загружается и активизируется при помощи функции DialogBox. Возврат из этой функции не производится, пока в ответ на какое-либо событие в диалоговой процедуре не будет вызвана функция EndDialog, т.е. функция DialogBox активизирует модальный диалог.

function DialogBox(hInstance: HINST; lpTemplate: PChar;
hWndParent: HWND; lpDialogFunc: pointer): Integer;

lpTemplate — Z-строка, содержащая имя ресурса диалога, hWndParent — хэндл окна-собственника диалога (обычно — главного окна программы, хотя можно указать 0), lpDialog-Function — указатель на процедуру диалога. Возвращаемое значение определяется программистом при вызове EndDialog внутри процедуры диалога.

function EndDialog(hDlg: HWND; nResult: Integer): BOOL;

hDlg — хэндл окна диалога (передается в процедуру диалога в качестве параметра), nResult — код завершения диалога, возвращаемый в вызывающую диалог программу. Определены некоторые стандартные коды, например ID_OK, ID_CANCEL, ID_ABORT, ID_IGNORE, ID_YES, ID_NO, ID_CLOSE. Диалог должен завершаться явным вызовом EndDialog в ответ на нажатие некоторой кнопки (обычно это Ok и Cancel), а также при получении диалоговой процедурой сообщения WM_CLOSE.

Существует возможность работать с немодальными диалоговыми окнами, используя для их создания функцию CreateDialog, однако этот вопрос здесь рассматриваться не будет.

Библиотеки DLL

DLL (Dynamic-Link Library, динамически связываемая библиотека) – это файл, содержащий набор готовых к исполнению процедур, а также, возможно, ресурсы. Функции DLL могут вызываться исполняемыми модулями (.EXE) или функциями других DLL, причем одновременно к функциям DLL может обращаться неограниченное количество других модулей.

Термин "динамическое связывание" означает, что функции библиотеки не включаются в код использующей ее программы, а вызываются из отдельно существующей библиотеки в процессе работы программы.

Механизм DLL является основой Windows: все функции API собраны в DLL-библиотеках, и исключительно благодаря поддержке механизма DLL программы способны общаться с операционной системой вообще. Написание собственной DLL и размещение ее в общедоступном каталоге можно с полным основанием считать расширением функций ОС.

Подключение DLL может осуществляться как автоматически на этапе загрузки программы (load-time linking, статическая загрузка), так и "вручную" во время работы программы посредством ряда функций API (runtime linking, динамическая загрузка). Код библиотеки загружается в память один раз, сколько бы процессов ею ни пользовались, и выгружается, когда все использующие процессы завершаются. Принудительное аварийное завершение процесса, загрузившего DLL, часто приводит к тому, что код DLL не выгружается и бесполезно занимает память.

DLL могут быть написаны на разных языках с использованием различных моделей вызова. В документации к DLL обязательно должен быть отражен момент, связанный с моделью вызова функций (правилами передачи параметров). Наиболее часто используемые модели вызова, поддерживаемые большинством компиляторов – это модели, обозначаемые в Delphi как **StdCall** (модель вызова функций WinAPI), **CDecl** (модель языка C) и **Pascal** (модель языка Паскаль).

Синтаксис исходного текста DLL

Исходный текст библиотеки DLL на языке Borland Pascal 8, используемом Delphi, чем-то напоминает исходный текст модуля (Unit) и выглядит следующим образом:

```
library <имя>;  // вместо Program или Unit
uses <обычный список используемых модулей>;

  <Объявление процедур и функций>

exports
  Proc1 index 10,
  Proc2 name 'ProcedureTwo',
  Proc3,
  .....;


begin
  <возможные начальные установки, напр. - для глоб. переменных>
end.
```

Стандартным образом, как для исполнимого модуля .EXE, в теле программы или в unit'ах описываются процедуры и данные.

Во фразе **exports** через запятую перечисляются идентификаторы экспортируемых, т.е. предназначенных для вызова внешними программами, процедур. Фраза **exports** может встречаться в тексте сколько угодно раз и перемежаться с объявлением процедур, единственное условие – процедура должна сначала объявляться, а затем экспортироваться. Рекомендуется объявлять экспортируемые процедуры с моделью вызова **StdCall**, гарантированно поддерживаемой трансляторами всех языков программирования для Windows. Экспортировать можно и процедуры из модулей.

Для каждой экспортируемой процедуры может быть в явном виде указан числовой идентификатор (компилятор по умолчанию присваивает последовательные числа) и символьный идентификатор (по умолчанию берется идентификатор из программы и переводится в верхний регистр). Именно по этим идентификаторам использующие DLL программы обращаются к ее функциям. В документации Microsoft рекомендуется пользоваться исключительно символьными идентификаторами, хотя это несколько замедляет процесс загрузки DLL.

Глобальные переменные DLL, в отличие от модуля (Unit), не могут быть экспортированы. Кроме того, если DLL используется в нескольких процессах, то для каждого процесса создается отдельная копия глобальных переменных DLL, и поэтому глобальные переменные DLL в принципе не могут быть использованы для передачи данных между процессами даже в случае доступа к ним через функции самой DLL.

DLL может содержать ресурсы, для их подключения используется все та же директива компилятора {$R}. Так как DLL является законченным исполнимым файлом, в отличие от модуля Unit, то ресурсы физически включаются в тело DLL.

Поиск используемых DLL

При использовании DLL может быть указан маршрут к ее файлу. Если маршрут указан, то файл должен находиться в указанной директории, иначе он не будет найден. Если же маршрут не указан (только имя файла), то поиск производится в следующем порядке:

1. Директория, откуда была загружена использующая DLL программа.
2. Текущая директория.
3. Директория SYSTEM32 операционной системы Windows 98/NT.
4. Директория SYSTEM операционной системы Windows.
5. Директория, куда установлены сами Windows.

6. Директории, перечисленные в переменной окружения PATH (обычно настраивается в файле autoexec.bat).

Подключение DLL на этапе загрузки

Для автоматического подключения функций DLL они объявляются в использующей программе как внешние при помощи директивы **external**. Параметры, естественно, должны быть описаны тоже. Например, если рассматриваемая в предыдущем разделе DLL скомпилирована в TESTLIB.DLL, а три процедуры, упомянутые в **export** имеют соответственно один, два и три целых 32-разрядных параметра и модель вызова **StdCall**, то они могут импортироваться следующим образом:

```
const testlib='testlib.dll';
procedure FirstProc(a:integer); external testlib index 10; stdcall;
procedure SecondProc(a,b:integer); external testlib name 'ProcedureTwo'; stdcall;
procedure PROC3(a,b,c:integer); external testlib; stdcall;
```

Третья процедура в этом случае должна называться точно так же, как и в библиотеке, так как для связывания будет использовано имя 'PROC3'.

После такого объявления и загрузки программы будет автоматически загружена библиотека TESTLIB.DLL, и при вызове в программе функций FirstProc, SecondProc и PROC3 будут вызываться соответствующие функции DLL.

Часто при создании DLL удобно сразу описать отдельный интерфейсный модуль (Unit) для импорта ее функций, а затем просто подключать такой модуль к использующей программе в фразе **uses**. Примером такого интерфейсного модуля является модуль WINDOWS, используемый во всех программах Delphi для доступа к функциям API.

При невозможности подключить библиотеку на этапе загрузки использующая ее программа не запускается.

Подключение DLL на этапе выполнения

Такой способ подключения обычно используется в двух случаях:

- когда функции DLL используются только в течение определенного этапа работы программы (например – аутентификация пользователя при запуске);

- когда программе требуются ресурсы из DLL.

Для подключения DLL используются функции:

function LoadLibrary(lpLibFileName: PChar): HMODULE;

function LoadLibraryEx(lpLibFileName: PChar; hFile: THandle; dwFlags: DWORD): HMODULE;

В качестве первого параметра передается указатель на строку с именем файла исполнимого модуля. С помощью этих функций можно загружать как DLL, так и EXE-файлы. Параметр hFile второй функции не используется и должен быть равен нулю. В параметре dwFlags можно задать дополнительные параметры загрузки библиотеки, самый интересный из которых – флаг LOAD_LIBRARY_AS_DATAFILE, полезный при работе с ресурсами библиотеки. Возвращаемое значение – хэндл загруженного исполнимого модуля, применяемый для функций API, или 0 при неудачной загрузке.

После того, как библиотека больше не требуется, использующая программа должна освободить ее при помощи вызова функции

function FreeLibrary(module: hModule):boolean;

В качестве параметра передается хэндл модуля, полученный при загрузке библиотеки.

Для доступа к функциям динамически загружаемых DLL необходимо получать адрес точки входа каждой функции с помощью функции

function GetProcAddress(module: hModule; procName: pChar): pointer;

В качестве первого параметра передается хэндл модуля загруженной DLL, в качестве второго – указатель на символьный идентификатор процедуры или числовой идентификатор, преобразованный к типу "указатель", например

pointer(10); - число 10, преобразованное к типу указатель

Возвращаемое значение – указатель на точку входа в процедуру или **nil**, если требуемая процедура отсутствует.

Использовать эту процедуру в программах на Паскале следует совместно с переменными процедурного типа (с указателями на процедуру), например так:

```
var Proc: procedure(a,b:integer); //процедура с двумя целыми параметрами
    hModule:THandle;
.....................
hModule:=LoadLibrary('testlib.dll');
@Proc:=GetProcAddress(hmodule,'ProcedureTwo'); //настройка
proc(1,2);  // вызов
FreeLibrary(hModule);
```

Имеется также возможность узнать хэндл уже загруженной библиотеки – например, если она загружена статическим образом при старте программы. Для этого служит функция

function GetModuleHandle(filename:PChar):HMODULE;

В качестве параметра необходимо передать имя файла уже загруженной библиотеки.

Использование ресурсов из DLL

Чтобы воспользоваться ресурсом из DLL, необходимо получить хэндл модуля этой DLL при помощи LoadLibrary или GetModuleHandle, после чего передать этот хэндл в обычную процедуру загрузки ресурса (LoadBitmap, LoadCursor и т.д.). Иногда DLL используются как чистые библиотеки ресурсов, т.е. не содержат исполнимых кодов вообще.

Чтобы загрузить библиотеку исключительно с целью доступа к ее ресурсам, рекомендуется применять функцию LoadLibraryEx примерно следующим образом:

```
hModule:=LoadLibraryEx('testlib.dll',0, LOAD_LIBRARY_AS_DATAFILE);
hBmp:=LoadBitmap(hModule,'bitmap1');
.................
DeleteObject(hBmp);
FreeLibrary(hModule);
```

При этом не выполняется код входа в DLL, что ускоряет загрузку.

Задание

Выполняется при самостоятельной подготовке:

1. Изучить теоретическую часть описания ЛР и материал соответствующих лекций.

2. По материалам Справки (Help) изучить описание следующих функций Windows и связанных с ними структур данных:

Функции:

Выполняется в лаборатории:

1. Включить компьютер. Запустить систему Delphi 2.0.

2. Загрузить исходный текст примера LAB7.PAS, изучить логику работы программы.

3. Откомпилировать и запустить пример. Изучить поведение созданных окон.

4. Запустить программу Resource Workshop, открыть файл ресурсов RES7.RES, сохранить его в виде файла .RC, изучить синтаксис текстового описания ресурсов.

5. Написать и отладить программу по индивидуальному заданию (см. ниже). Продемонстрировать результаты работы преподавателю.

6. Завершить работу с Delphi и Resource Workshop. Оставить компьютер включенным.

Варианты заданий:

В этой работе необходимо написать программу, пользующуюся ресурсами и функциями из DLL. Главное окно программы должно быть снабжено меню и собственным нестандартным значком. В одном из окон программы должен использоваться нестандартный курсор мыши. Все строки, битовые образы, курсоры, значки, используемые в программе, должны быть описаны в ресурсах.

Один из пунктов меню должен активизировать модальный диалог из ресурса, поведение которого соответствует заданию на ЛР №6.

Другой пункт меню должен активизировать "окно-заставку", фон которого заполнен битовым образом способом, указанным в задании ЛР №4, предусмотреть вывод номера бригады в этом окне.

Выход из программы также предусмотреть при помощи соответствующего пункта меню.

Еще один пункт меню должен приводить к вызову функции готовой DLL с именем LIB7.DLL в соответствии с вариантом (см. таблицу). В качестве параметра PChar необходимо передавать указатель на существующую строку, заканчивающуюся нулем.

№	Идентификатор	Параметры	Возвр. значение	Модель вызова	Способ загрузки DLL
1	1	a: integer; b: integer;	нет	C	статический
2	2	a: pchar; b: pchar;	integer	Pascal	динамический
3	'proc3'	a: pchar; b: char;	boolean	API	статический
4	'proc4'	a: pchar; b: pchar;	pointer	C	динамический
5	5	a: pchar; var b: integer;	нет	Pascal	статический
6	6	a: integer; b: integer;	integer	API	динамический
7	'proc7'	a: pchar; b: pchar;	char	C	статический
8	'proc8'	a: pchar; b: char;	smallint	Pascal	динамический
9	9	a: pchar; b: pchar;	boolean	API	статический
10	'proc10'	a: pchar; var b: integer;	нет	C	динамический