# Урок 6. Ресурсы приложения и их использование

В большинство программ Windows включаются пиктограммы, меню и блоки диалога, различные курсоры. Все это виды ресурсов (resource) Windows.

## Файлы ресурсов приложения

С помощью файлов ресурсов большинство приложений определяют визуальные элементы своего пользовательского интерфейса: меню, диалоговые окна, надписи, растровые изображения и другие типы ресурсов.

Файлы ресурсов (rc-файлы) создаются в виде текстового файла (этот файл можно создать как текстовым редактором, так и специальным редактором ресурсов) и компилируются компилятором ресурсов. Полученный в результате двоичный файл (res-файл) обычно компонуется с оставшимися частями приложения, образуя единый двоичный образ, содержащий выполняемый программный код и информацию о ресурсах. На финальном этапе сборки загрузочного модуля компилятор ресурсов вызывается еще раз для записи ресурсов в загрузочный модуль. Также он формирует специальную таблицу ресурсов, расположенную в заголовке exe-файла. Таблица используется Windows для поиска и загрузки ресурсов в оперативную память.

Ресурсы являются данными, и они хранятся в exe-файле программы, но расположены они не в области данных, где обычно хранятся данные исполняемых программ. Таким образом, к ресурсам нет непосредственного доступа через переменные, определенные в исходном тексте программы. Они должны быть явно загружены из exe-файла в память.

**Замечание 1.** Использование файла ресурсов не является обязательным, однако, использование файла ресурсов обеспечивает очень простой способ замены зависящих от языка элементов пользовательского интерфейса при работе приложения в многоязыковой среде.

**Замечание 2.** Ресурсы не обязательно должны компоноваться с исполняемым файлом приложения, они также могут сводиться в отдельную библиотеку DLL. Преимущество этого подхода заключается в том, что при изменении файла ресурсов не требуется перекомпилировать все приложение, а нужно только заменить DLL (shells32.dll).

## Компоненты файла ресурсов

Файл ресурсов или сценарий ресурсов может содержать любое количество утверждений сценария ресурсов и директив препроцессора. Например, простой файл ресурсов:

```
#include <windows.h>
DlgBox DIALOG 20, 20, 200, 200
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "SampleDialog"
BEGIN
        DEFPUSHBUTTON "Sample Button", IDOK, 20, 45, 50, 15, WS_GROUP
        CTEXT "Sample Message", -1, 10, 10, 90, 8
END
```

Этот простой сценарий определяет диалоговое окно с одной кнопкой и одним статическим полем (с надписью). Из приложения, использующего такой ресурс, можно сослаться на это диалоговое окно по имени DlgBox.

Некоторые константы (например, IDOK, DS_MODALFRAME или WS_GROUP) определены в файле windows.h и не являются частью языка сценария ресурсов. Директива #include в стиле C/C++ используется для указания файла, содержащего макроопределения.

Обычно оператор файла ресурсов состоит из ***идентификатора,*** который может быть или текстовой строкой (надписью, как DlgBox в предыдущем примере), или числовым значением, за которым следует сам оператор. Оператор может состоять из одной строки (в таком случае за ним следует один или несколько параметров) или из многих строк (в этом случае за оператором следует блок инструкций).

Однострочные операторы определяют растровые изображения (BITMAP), курсоры (CURSOR), шрифты (FONT), пиктограммы (ICON), язык ресурсов (LANGUAGE). Однострочный оператор состоит из следующих частей:

1. Идентификатор  Тип_ресурса  [ Характеристика_загрузки ]  Имя_файла_ресурса
2. Операторы  BITMAP, CURSOR, FONT и ICON также принимают дополнительные параметры. Эти параметры указывают характеристики загрузки и памяти для ресурса. В Windows используется только один атрибут DISCARDABLE, указывающий, что ресурс может быть удален из памяти, если он больше не требуется, например:

```
MyBitmap BITMAP DISCARDABLE "mybitmap.bmp"
```

Оператор LANGUAGE устанавливает язык всех последующих ресурсов до конца сценария ресурсов или до следующего оператора LANGUAGE. Этот оператор также можно использовать для установки языка определенного ресурса в многострочном операторе, если он находится непосредственно перед ключевым словом BEGIN в таком операторе. После оператора LANGUAGE следуют идентификаторы языка, которые должны быть константами, определенными в файле winnls.h.

3. Многострочные операторы в файле ресурсов определяют диалоговые окна (DIALOG), таблицы строк (STRINGTABLE), таблицы акселераторов (ACCELERATORS), меню (MENU), пользовательские данные (RCDATA) и ресурсы информации о версии. Многострочные операторы начинаются с идентификатора, оператора и необязательных параметров, затем следует блок инструкций, заключенный между ключевыми словами BEGIN-END:

```
Идентификатор ОПЕРАТОР [Необязательные_параметры]
[Необязательные_инструкции]
BEGIN
        [Инструкции]
END
```

Необязательные_инструкции могут включать, например, инструкцию CAPTION, которая определяет заголовок диалогового окна, или инструкцию STYLE для установки стилей диалогового окна.

## Пиктограммы

В приложение можно добавить ресурс, который называется пиктограммой (icon). Пиктограмма - это графическое изображение небольшого размера, состоящее из отдельных пикселов (32 x 32).

Пиктограммы обычно используют для обозначения свернутого окна приложения, они отображаются в левом верхнем углу заголовка окна (размером 16x16, из “большого” изображения исключается каждый второй столбец и строка). С помощью редактора изображений можно нарисовать свои пиктограммы и использовать их в приложениях.

Приложения активно работают с графическими изображениями, состоящими из отдельных пикселов и имеющих прямоугольную форму. Такие изображения называются bitmap (битовый образ). Можно считать, что пиктограмма является упрощенным вариантом изображения bitmap.

Пиктограммы хранятся в файлах с расширением имени *.ico. В одном файле обычно находится несколько вариантов пиктограмм. Когда Windows рисует пиктограмму, она выбирает из файла наиболее подходящую для текущего режима работы видеоадаптера.

Для включения пиктограммы в файл описания ресурсов используется оператор ICON, ссылка на файл пиктограммы выглядит примерно так (в этом операторе пиктограмме присваивается имя “MyIcon”):

```
MyIcon ICON iconfile.ico
```

В приложении для получения дескриптора пиктограммы используется функция LoadIcon:

```
HICON LoadIcon(HINSTANCE hInst, LPCTSTR lpszIcon);
```

Первым параметром является дескриптор экземпляра приложения. Он необходим для того, чтобы Windows могла определить, в каком файле содержится ресурс пиктограммы.

Вторым параметром является имя пиктограммы из описания ресурсов, заданное в виде указателя на строку.

Если первый параметр функции LoadIcon установлен в NULL, то Windows узнает, что та пиктограмма является предопределенной (с префиксом IDI_):

```
//загрузка предопределенной пиктограммы IDI_APPLICATION:
LoadIcon(NULL, IDI_APPLICATION);
```

Рассмотрим способы загрузки пиктограмм, определенных в ресурсах приложения.

Имеется связь между именем пиктограммы в описании ресурсов и в вызове функции LoadIcon:

| | |
|---|---|
| Описание ресурсов: | MyIcon ICON iconfile.ico |
| Текст программы: | HICON hIcon = LoadIcon(hInst, "MyIcon"); |

Вместо имени можно также использовать **число** (16-разрядное беззнаковое WORD) – это число называется **идентификатором пиктограммы**. Тогда связь между ссылкой на пиктограмму в описании ресурсов и в вызове функции LoadIcon имеет следующий вид:

| | |
|---|---|
| Описание ресурсов: | 125 ICON iconfile.ico |
| Текст программы: | HICON hIcon = LoadIcon(hInst, MAKEINTRESOURCE(125)); |

MAKEINTRESOURCE является макросом, определенным в заголовочных файлах Windows, который преобразует число в указатель, но со старшими разрядами, установленными в нуль. Так Windows определяет, что второй параметр функции LoadIcon является числом, а не указателем на символьную строку.

На имя пиктограммы можно ссылаться и с помощью третьего метода, который является комбинацией первых двух методов (по символу # операционная система определяет, что далее следует число в ASCII-коде):

| | |
|---|---|
| Описание ресурсов: | 125 ICON iconfile.ico |
| Текст программы: | HICON hIcon = LoadIcon(hInst, ”#125”); |

Есть еще и четвертый способ установки связи между ссылкой на пиктограмму в описании ресурсов и в вызове функции LoadIcon. В этом способе используется макроопределение, которое должно включаться и в файл описания ресурсов, и в исходный текст программы:

| | |
|---|---|
| Заголовочный файл: | #define MyIcon 125 |
| Описание ресурсов: | MyIcon ICON iconfile.ico |
| Текст программы: | HICON hIcon = LoadIcon(hInst,<br>MAKEINTRESOURCE(MyIcon)); |

Использование идентификаторов вместо имен пиктограмм уменьшает размер exe-файла, и вероятно, немного ускоряет работу функции LoadIcon.

Дескриптор, выдаваемый функцией LoadIcon, можно присвоить полю пиктограммы структуры класса при регистрации класса окна hWnd, тогда окно hWnd будет иметь именно эту пиктограмму:

```
wndclass.hIcon = LoadIcon(hInst, "MyIcon");
```

Для того, чтобы в любой момент переопределить значение поля пиктограммы в структуре класса окна hWnd, следует воспользоваться вызовом:

```
SetClassLong(hWnd, GCL_HICON, LoadIcon(hInst, "AnotherIcon"));
```

**Замечание**. После того, как приложение загрузило пиктограмму, ее можно также нарисовать в любом месте экрана функцией DrawIcon.

Если загруженная с помощью функции LoadIcon пиктограмма приложению больше не нужна, то можно освободить занимаемую ею память, вызвав функцию DestroyIcon.

## Курсоры

Приложение может использовать не только стандартные курсоры, но и альтернативные, хранящиеся в ресурсах приложения. Курсор мыши представляет собой не что иное, как упрощенный вариант битового изображения, аналогично пиктограмме.

Инструкции для задания курсора в файле описания ресурсов и для получения его дескриптора в программе очень похожи на аналогичные инструкции для пиктограмм:

| | |
|---|---|
| Описание ресурсов: | MyCursor CURSOR cursfile.cur |
| Текст программы: | HCURSOR hCursor = LoadCursor(hInst, "MyCursor"); |

Другие способы, показанные для пиктограмм (использование идентификаторов и MAKEINTRESOURCE), также работают и для курсоров.

Дескриптор курсора мыши, полученный при вызове функции LoadCursor, можно использовать при задании поля курсора структуры класса при регистрации класса окна. Это заставляет курсор мыши, если он оказывается в рабочей области окна, превращаться в пользовательский курсор:

```
wndclass.hCursor = LoadCursor(hInst, "MyCursor");
```

Изменить курсор для окна можно в любое другое время, переопределив значение, содержащееся в структуре класса окна:

```
SetClassLong(hWnd, GCL_HCURSOR, LoadCursor(hInst, "AnotherCursor"));
```

**Замечание**. Для динамического изменения формы курсора следует использовать функцию SetCursor, используя в качестве параметра дескриптор нового курсора, подготовленный при помощи функции LoadCursor.

Функцию SetCursor следует вызывать при обработке сообщения WM_MOUSEMOVE.

В противном случае для перерисовки курсора при его движении Windows использует курсор, ранее заданный в классе окна.

**Замечание**. Для того чтобы выключить изображение курсора или вновь его включить, используют функцию ShowCursor.

Операционная оболочка Windows содержит несколько встроенных курсоров. Их идентификаторы описаны в файле windows.h и начинаются с префикса IDC_.

Для загрузки встроенных курсоров также используется функция LoadCursor, но в качестве первого параметра ей передается NULL:

```
LoadCursor(NULL, IDC_ARROW);
```

Если курсор, загруженный с помощью функции LoadCursor, приложению больше не нужен, то можно освободить занимаемую им память, вызвав функцию DestroyCursor.

## Битовые изображения

Битовые образы используются в двух главных целях.

**Первая** – рисование на экране картинок: битовые образы можно просто рисовать в рабочей области окна; можно использовать их в кнопках, определяемых пользователем; применять при работе с меню, пункты которого отрисовываются пользователем.

**Вторая** цель использования битовых образов – создание кистей. Кисти являются шаблонами пикселей, которые Windows использует для закрашивания изображаемых на экране областей.

Одно из простых применений изображений bitmap - раскрашивание фона окна. После загрузки изображения функцией LoadBitmap приложение должно создать из этого изображения кисть, вызвав функцию CreatePatternBrush.

В ресурсы приложения можно включать произвольные графические изображения в виде битового образа (изображение типа bitmap). С помощью графических редакторов можно нари-

совать графическое изображение типа bitmap. Изображение записывается в файл с расширением имени .bmp.

Изображения bitmap удобно использовать для внешнего вида окон приложения, они открывают широкие возможности для разработки дизайна приложения Windows.

Битовый образ включается в файл описания ресурсов в том же формате, что значок или курсор:

```
MyBmp BITMAP bmpfile.bmp
```

Для загрузки ресурса используется функция LoadBitmap. Ее возвращаемым значение является дескриптор битового образа:

```
HBITMAP hBitmap = LoadBitmap(hInst, "MyBmp");
```

Другие способы, показанные для пиктограмм (использование идентификаторов и MAKEINTRESOURCE), также работают и для битовых изображений.

Функция CreatePatternBrush возвращает идентификатор кисти, который можно использовать при регистрации окна (перед завершением работы приложения созданная кисть должна быть уничтожена функцией DeleteObject):

```
HBRUSH hBrush = CreatePatternBrush(hBitmap);
wndclass.hbrBackground = hBrush;
```

Когда Windows раскрашивает этой кистью область экрана, битовый образ повторяется по горизонтали и вертикали через каждые восемь пикселей.

После загрузки растрового изображения его можно также использовать для вывода в рабочую область окна.

Рассмотрим пример вывода битового изображения при ответе на сообщение WM_PAINT. В приводимом примере размер выводимого битового изображения 60x60 пикселя; выводится оно, начиная от точки с координатами (100, 100):

```
PAINTSTRUCT ps;
HDC hDC = BeginPaint(hWnd, &ps);
HDC BitmapDC = CreateCompatibleDC(hDC);
HBITMAP hOldBitmap = SelectBitmap(BitmapDC, hBitmap);
BitBlt(hDC, 100, 100, 60, 60, BitmapDC, 0, 0, SRCCOPY);
SelectBitmap(BitmapDC, hOldBitmap);
DeleteDC(BitmapDC);
EndPaint(hWnd, &ps);
```

Этот код создает еще один контекст устройства, совместимый с контекстом устройства окна. Это делается при помощи функции CreateCompatibleDC. Прежде чем растровое изображение сможет быть выведено на экран, оно сначала должно быть выбрано в этот второй контекст устройства.

Функция SelectObject выбирает объект в данный контекст устройства. Она всегда возвращает объект, который перед этим был в контексте устройства (для последующего восстановления). После того, как растровое изображение показано и восстановлено предыдущее растровое изображение, нужно удалить второй контекст устройства при помощи DeleteDC.

Главное отличие между битовыми образами и остальными ресурсами в их практической важности: битовые образы являются объектами GDI.

Хороший стиль составления программ рекомендует, что битовые образы, созданные функцией LoadBitmap, должны быть удалены при помощи функции DeleteObject, как только в них пропадает необходимость или если программа завершается.

## Символьные строки

Ресурсы символьных строк предназначены для облегчения перевода приложения на другие языки.

Ресурс таблицы строк определяет произвольное количество текстовых строк. Приложения ссылаются на эти строки по символьным идентификаторам.

Основное преимущество использования таблицы строк – размещение всех зависимых от языка компонентов в файле ресурсов, что существенно упрощает задачу локализации приложения.

Таблица строк определяется ключевым словом STRINGTABLE, за которым указаны необязательные инструкции и одно или более определений строк, заключенных между ключевыми словами BEGIN-END:

```
STRINGTABLE
BEGIN
        id1     "character string 1"
        id2     "character string 2"
        // определения остальных строк
END
```

Идентификаторы строк, которые предшествуют каждой строке, должны быть или числами, или идентификаторами макроопределений, которые задаются в заголовочном файле. В описании ресурсов может быть только одна таблица строк. Максимальный размер каждой строки – 255 символов.

Например:

```
STRINGTABLE
BEGIN
        IDS_HELLO       "Hello"
        IDS_BYE         "Good bye"
END
```

Здесь IDS_HELLO и IDS_BYE – символьные идентификаторы, определенные в другом месте, например, так:

```
#define IDS_HELLO       1
#define IDS_BYE         2
```

Для копирования строки из ресурсов приложения в буфер в сегменте данных приложения, следует использовать функцию LoadString, затем этот буфер можно использовать как обычную строку:

```
char szBuffer[256];
LoadString(hInst, id, szBuffer, iMaxLength);
MessageBox(hWnd, szBuffer, "Text from stringtable", MB_OK);
```

Параметр id соответствует идентификатору, который предшествует каждой строке в файле описания ресурсов; iMaxLength – максимальное число передаваемых в szBuffer символов.

## Ресурсы, определяемые пользователем

Синтаксис файла ресурсов позволяет создавать также пользовательские типы ресурсов.

Ресурсы, определяемые пользователем (user-defined resource) удобны для включения самых разнообразных данных в загрузочный файл и для получения доступа в приложении к этим данным. Данные могут содержаться в любом формате – текстовом или бинарном.

При загрузке данных в оперативную память возвращаемым значением функций Windows, которые используются для доступа к определяемым пользователем ресурсам, является ***указатель*** на данные. С этими данными можно делать все, что угодно приложению.

Операторы для пользовательских ресурсов могут быть как однострочными, так и многострочными.

## Процесс использования многострочных операторов для включения пользовательских ресурсов

Однострочные операторы используются для указания пользовательских ресурсов, которые хранятся в отдельных файлах. Вот как, например, в файле ресурсов приложения описывается ссылка на файл с данными:

```
MyData TEXT filedata.dat
```

Имена MyData (имя ресурса) и TEXT (тип ресурса) в операторе могут быть любыми. В данном примере был создан ресурс собственного типа, который называется TEXT.

Для того, чтобы получить дескриптор ресурса, необходимо вызвать функцию LoadResource:

```
HGLOBAL hResource = LoadResource(hInst, FindResource(hInst, "TEXT", "MyData"));
```

Вместо имен и типов в описании пользовательского ресурса можно использовать числа. Числа могут быть преобразованы в указатели при вызове функции FindResource с использованием MAKEINTRESOURCE. Числа, используемые в качестве типа ресурса, должны быть больше 255 (меньшие числа использует Windows).

Несмотря на свое имя, функция LoadResource фактически не загружает ресурс сразу в оперативную память. Когда к ресурсу необходимо получить доступ, следует вызвать функцию LockResource, используя оператор такого типа:

```
ТИП *pt r = (ТИП *)LockResource( hResource );
```

Функция LockResource загружает ресурс в память (если он еще не был загружен), и возвращает указатель на него.

После окончания работы с этим ресурсом приложение может освободить оперативную память, вызвав функцию FreeResource:

```
FreeResource(hResource);
```

## Процесс использования многострочных операторов для включения пользовательских ресурсов

Многострочные операторы для пользовательских ресурсов используются для внедрения определений пользовательских ресурсов в файл ресурсов. Синтаксис многострочных операторов пользовательских ресурсов:

```
имя тип [опции]
BEGIN
        неструктурированные_данные
END
```

Блок неструктурированные_данные может содержать целые значение в десятичном, шестнадцатеричном или восьмеричном представлении или строки, заключенные в двойные кавычки. Строки неструктурированных данных должны явно завершаться нулем. Отдельные пункты данных разделяются запятыми.

Неструктурированные данные также можно определить в форме ресурса RCDATA. Синтаксис RCDATA и синтаксис многострочного оператора почти идентичны, но в случае оператора неструктурированных данных вместо типа используется ключевое слово RCDATA, а также этот оператор может содержать необязательные операторы CHARACTERISTICS, LANGAGE и VERSION.

Например:

```
resname RCDATA
BEGIN
"Here is an ANSI string\0",    // строка, ограниченная нулем */
L"Here is a Unicode string\0", // строка UNICODE, ограниченная нулем
1024,                          // целое, хранящееся WORD
7L,                            // целое, хранящееся DWORD
0x029a,                        // шестнадцатеричное целое
0733,                          // восьмеричное
END
```