7. Разбор функций создания основных графический объектов программы

(окна, иконки, курсоры, фон). Назначение различных типов окон.

План.

1. LoadIcon.

2. LoadCursor.

3. GetStockObject.

4. CreateWindowEx.

5. Стили окон.

На прошлой лекции мы написали первую работающую программу, в которой

используется несколько функций WinAPI (Windows Application

Programming Interface - интерфейс прикладного программирования, а

проще - стандартная библиотека функций Windows). Но мы не разобрали,

для чего предназначены эти функции: какие параметры получают, что

делают и возвращают. Этим мы сегодня и займемся.

Первой такой функцией будет LoadIcon. Вот ее прототип:

HICON LoadIcon(HINSTANCE hInst, LPCTSTR lpIconName);

Эта функция загружает иконку в память и возвращает ее дескриптор.

HIOCN - это еще один тип дескрипторов, на этот раз иконок. Теперь

рассмотрим параметры, которые получает эта функция:

hInst - дескриптор приложения, в ресурсах которого надо искать иконку.

Забегая вперед, скажу, что в файл приложения - exe-шник - можно

включать различные ресурсы: картинки, иконки, курсоры и т.д.. Среди

этих ресурсов и будет искаться изображения иконки.

lpIconName - строка, содержащая имя ресурса иконки в указанном

приложении. По этому имени и будет искаться иконка.

Но т.к. мы не создавали никаких ресурсов, то и файл нашего приложения

не содержит никаких иконок. Поэтому мы используем одну из стандартных

иконок. Для этого в качестве дескриптора приложения надо указать NULL,

а вместо указателя на строку - одну из констант:

константа что будет изображено на иконке

IDI\_APPLICATION прямоугольное окно

IDI\_ASTERISK i (информация)

IDI\_EXCLAMATION ! (восклицательный знак)

IDI\_HAND рука

IDI\_QUESTION ? (вопросительный знак)

IDI\_WINLOGO флажок Windows

(префикс IDI расшифровывается - ID Icon)

В нашей программе эта функция используется дважды:

winclass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

winclass.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

В первой строчке загружается большая иконка (она используется для

изображения в списке при нажатии Alt-Tab), во второй - маленькая

иконка (изображается в левом верхнем углу окна).

Следующей рассмотренной функцией будет LoadCursor. Эта функция

практически аналогична предыдущей. Прототип:

HCURSOR LoadCursor(HINSTANCE hInst, LPCTSTR lpCursorName);

Здесь используется еще один тип дескрипторов - HCURSOR. Это

дескриптор курсора. Входные параметры функции совпадают по назначению

с параметрами предыдущей функции. Т.е. это дескриптор приложения и

строка имени ресурса. И также есть возможность использовать

стандартные курсоры:

константа какой будет курсор

IDC\_ARROW стандартная стрелка

IDC\_APPSTARTING стрелка с маленькими часами

IDC\_CROSS крест

IDC\_NO перечеркнутый круг

IDC\_SIZEALL курсор перемещения (4 стрелки)

IDC\_WAIT песочные часы

(префикс IDC расшифровывается - ID Cursor)

Я указываю только самые часто используемые константы. Остальные можно

найти в стандартных библиотеках.

А вот строчка с использованием этой функции из программы :

winclass.hCursor = LoadCursor(NULL, IDC\_ARROW);

Стандартные курсоры используются гораздо чаще, чем стандартные иконки.

Это происходит по понятным причинам. При создании новой программы

иконка используется как некоторый символ, являющийся ее лицом, по

которому ее отличают от всех других. Поэтому она должна быть

уникальной. Курсор же является частью инструментария, и, если он

выполняет стандартные действия, то и изображаться должен стандартно.

Новый курсор следует создавать только в том случае, если действия, им

совершаемые, являются специфическими (например, в графических

программах).

Теперь рассмотрим функцию GetStockObject. Она возвращает дескрипторы

еще двух объектов: перьев и кистей. Перья используются для рисования

рамок графических объектов, а кисти для заливки графических объектов.

Прототип этой функции такой:

HANDLE GetStockObject(int uType);

Есть функции для создания перьев и кистей любого из 24 миллиона

цветов, но не всегда это необходимо. Иногда можно ограничиться

использованием одного из стандартных цветов. Так вышло, что Windows

использует их в большинстве случаев. Но тогда вместо того, чтобы

каждый раз создавать эти кисти, можно один раз создать, а потом

использовать по мере надобности. Так все и работает: Windows создает

кисти и перья стандартных цветов, а функция GetStockObject возвращает

дескрипторы этих объектов.

Итак, параметр у нее только один: uType - константа, определяющая,

дескриптор какого объекта возвращать. Она может принимать следующие

значения:

BLACK\_BRUSH черная кисть

WHITE\_BRUSH белая кисть

GRAY\_BRUSH серая кисть

NULL\_BRUSH прозрачная кисть (фон окна будет прозрачным)

BLACK\_PEN черное перо

WHITE\_PEN белое перо

NULL\_PEN прозрачное перо

и т.д.

Т.к. возвращаемые объекты могут быть разные (перья и кисти), то

возвращаемое значение является типом HANDLE - общим типом для всех

дескрипторов. Поэтому после вызова этой функции необходимо делать

приведение типа к HPEN или HBRUSH соответственно. Вот как это

делается в нашей программе:

winclass.hbrBackground = (HBRUSH) GetStockObject(BLACK\_BRUSH);

Прежде чем записать результат работы функции GetStockObject в

переменную winclass.hbrBackground его тип меняется на HBRUSH.

Это делает запись (HBRUSH) перед функцией. Таким же образом можно

делать практически любые преобразования типов, но следует быть

осторожнее - ошибки в таких и от таких преобразований очень сложно

отыскать.

Последняя на сегодня функция будет CreateWindowEx. Эта функция создает

окно, основываясь на 12 параметров. Вот ее прототип:

HWND CreateWindowEx(DWORD dwExStyle,

LPCTSTR lpClassName,

LPCTSTR lpWindowName,

DWORD dwStyle,

int x,

int y,

int nWidth,

int nHeight,

HWND hWndParent,

HWND hMenu,

HINSTANCE hInst,

LPVOID lpParam);

dwExStyle - расширенный стиль окна, используется для создания

стандартных типов окон: кнопки, edit-ы, combobox-ы и т.д.. В нашем

случае - NULL.

lpClassName - строка, содержащая название класса окна. Здесь нужно

указать название нашего зарегистрированного класса.

lpWindowName - строка, содержащая заголовок окна.

dwStyle - стиль изображения окна. Это комбинация нескольких флагов

из списка:

\* WS\_OVERLAPPED перекрываемое окно с рамкой

WS\_VISIBLE изначально видимое

\* WS\_SYSMENU есть кнопка системного меню

WS\_BORDER в тонкой рамке

\* WS\_CAPTION есть строка заголовка

WS\_ICONIC минимизированное окно

WS\_MINIMAZE минимизированное окно (то же, что и WS\_ICONIC)

\* WS\_MINIMAZEBOX есть кнопка минимизации

WS\_MAXIMAZE максимизированное окно

\* WS\_MAXIMIZEBOX есть кнопка максимизации

WS\_SIZEBOX с изменяемым за рамку размером

На самом деле почти всегда используются несколько из этих флагов.

Чтобы не набирать их каждый раз есть константа, содержащая их. Это

WS\_OVERLAPPEDWINDOW - в нее входят флаги, помеченные звездочками.

Этот набор флагов - стандартный набор для большинства окон.

x и y - координаты левого верхнего угла окна на экране в пикселах.

Если вместо них указать константу CW\_USEDEFAULT, то для создания

Windows сама их придумает (она обычно распологает окна лесенкой).

nWidth и nHeight - ширина и высота окна в пикселах. Тоже можно

использовать константу CW\_USEDEFAULT.

hWndParent - дескриптор окна родителя для данного окна. Дело в том,

что у каждого окна может быть родитель. Тогда, например, при закрытии

окна родителя закрываются все его дети. Есть и много других

особенностей, которые нам сейчас не нужны. Наше окно создается первым,

а поэтому у него не может быть родителя. В этом случае надо в этом

параметре передать NULL.

hMenu - дескриптор меню. Меню тоже является окном и, если мы бы хотели,

чтобы у нашего окна было меню, надо было бы передать в этом параметре

его дескриптор. У нас меню нет, поэтому передаем NULL.

hInst - дескриптор приложения, к которому принадлежит окно.

Обязательный параметр. Надо передать дескриптор запущенного приложения.

lpParam - указатель на данные создания окна. Этот параметр был сделан

на всякий случай и применения ему пока не нашлось. Поэтому всегда надо

передавать NULL.

А теперь использование этой функции в нашей программе:

hWnd = CreateWindowEx(NULL,

WINDOW\_CLASS\_NAME,

"My Window",

WS\_OVERLAPPEDWINDOW | WS\_VISIBLE,

0, 0, 400, 400,

NULL, NULL, hInst, NULL);

Здесь создается окно класса WINDOW\_CLASS\_NAME с заголовком "My Window",

со стандартными кнопками и изначально видимое, в месте (0, 0),

ширина 400, высота 400, родителя не имеет, меню не имеет, принадлежит

к приложению hInst.

[<< Назад](http://diabalda.narod.ru/kuznecov/lect6.htm) [Вернуться к списку](http://diabalda.narod.ru/kuznecov/index.htm) [Дальше >>](http://diabalda.narod.ru/kuznecov/lect8.htm)