https://nestor.minsk.by/kg/2008/20/kg82019.html

[https://nestor.minsk.by/kg/logo.jpg](https://nestor.minsk.by/kg/)

[](https://nestorclub.com/index.pl?act=NEWSSHOW&id=2020120301)

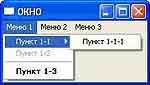
**Ассемблер под Windows для чайников. Часть 5**

[главная](https://nestor.minsk.by/kg/) » [статьи](https://nestor.minsk.by/kg/arch.html) » [2008](https://nestor.minsk.by/kg/2008/) » [21](https://nestor.minsk.by/kg/2008/21/) » 29.05.2008‣программирование

**Итак, основная задача на сегодня — меню. Второстепенные задачи будем привинчивать к основному коду. Меню — это достаточно важный компонент окна. Просмотрев пункты меню какой-нибудь новой или малоизвестной программы, пользователь должен иметь возможность получить общее представление о назначении программы и ее функционале. Чтобы меню было интуитивно понятно пользователю, программисту необходимо придерживаться негласного стандарта расположения элементов меню. Это значит, что сначала идут пункты Файл, Правка, Вид, а в конце — Справка.**

Меню является одним из типов ресурсов. Бывают такие ресурсы, как иконки, картинки bmp, курсоры и т.д. Традиционно ресурсы описываются в отдельном файле с расширением .rc и компилируются в файл .res компилятором ресурсов. Компилятор FASM нарушает традиции предков и позволяет размещать описание ресурсов прямо в исходном коде — в отдельной секции ресурсов. Для правильного отображения русских символов из ресурсов необходимо подключать макрос 'encoding\WIN1251.INC' (Code Page 1251 — это стандартная 8-битная кодировка для русских версий windows). На мой взгляд, очень удобно описывать ресурсы прямо в тексте программы, однако бывают ситуации, когда гораздо проще и быстрее подключить уже готовый res-файл к тексту программы — у вас есть полное на это право и возможность. Для этого используется директива вида: **section '.rsrc' resource from 'resfile.res' data readable**, где .rsrc — это название секции, а resfile.res — скомпилированный файл ресурсов. Теперь извольте ознакомиться с примером, в котором ресурсы описаны прямо в тексте программы:  
  
**format PE GUI 4.0  
entry start**

**include 'win32a.inc'  
include 'encoding\WIN1251.INC'  
  
section '.data' data readable writeable  
  
class db 'FASMWIN32',0  
title db 'ОКНО',0  
hwnd dd ?  
  
wc WNDCLASS 0,WindowProc,0,0,0,0,0,COLOR\_BTNFACE+1,0,class  
  
msg MSG  
  
section '.code' code readable executable  
  
start:  
invoke GetModuleHandle,0  
mov [wc.hInstance],eax  
invoke LoadIcon,0,IDI\_APPLICATION  
mov [wc.hIcon],eax  
invoke LoadCursor,0,IDC\_ARROW  
mov [wc.hCursor],eax  
invoke RegisterClass,wc  
cmp eax,0  
je error  
invoke LoadMenu,[wc.hInstance],1  
invoke CreateWindowEx,0,class,title,WS\_VISIBLE+WS\_OVERLAPPEDWINDOW,\  
CW\_USEDEFAULT,CW\_USEDEFAULT,CW\_USEDEFAULT,CW\_USEDEFAULT,\  
0,eax,[wc.hInstance],0  
cmp eax,0  
je error  
mov [hwnd],eax  
msg\_loop:  
invoke GetMessage,msg,0,0,0  
cmp eax,0  
je end\_loop  
invoke IsDialogMessage,[hwnd],msg  
cmp eax,0  
jne msg\_loop  
invoke TranslateMessage,msg  
invoke DispatchMessage,msg  
jmp msg\_loop  
  
error:  
invoke MessageBox,0,0,0,0  
  
end\_loop:  
invoke ExitProcess,[msg.wParam]  
  
proc WindowProc hwnd,wmsg,wparam,lparam  
push ebx esi edi  
cmp [wmsg],WM\_DESTROY  
je .wmdestroy  
.defwndproc:  
invoke DefWindowProc,[hwnd],[wmsg],[wparam],[lparam]  
jmp .finish  
.wmdestroy:  
invoke PostQuitMessage,0  
mov eax,0  
.finish:  
pop edi esi ebx  
ret  
endp  
  
section '.idata' import data readable writeable  
  
library kernel32,'KERNEL32.DLL',\  
user32,'USER32.DLL'  
  
include 'api\kernel32.inc'  
include 'api\user32.inc'  
  
section '.rsrc' resource data readable  
directory RT\_MENU,menus  
  
resource menus,\  
1,LANG\_RUSSIAN+SUBLANG\_DEFAULT,main\_menu  
  
menu main\_menu  
menuitem 'Меню 1',10,MFR\_POPUP  
menuitem 'Пункт 1-1',11,MFR\_POPUP  
menuitem 'Пункт 1-1-1',11,MFR\_END  
menuitem 'Пункт 1-2',12,MFT\_STRING,MFS\_GRAYED  
menuseparator  
menuitem 'Пункт 1-3',13,MFR\_END,MFS\_DEFAULT  
menuitem 'Меню 2',20,MFR\_POPUP  
menuitem 'Пункт 2-1',21,MFR\_END,MFS\_CHECKED  
menuitem 'Меню 3',30,MFR\_END**



Окно у вас должно получиться большего размера, чем на рисунке. Просто для экономии места я его уменьшил перед фотографированием. Координаты расположения и размеры окна в данном примере установлены в CW\_USEDEFAULT. Это значит, что система установит значения по умолчанию для координат и размеров окна. Константа CW\_USEDEFAULT может применяться только к окнам верхнего уровня (overlapped windows). Если применить ее к дочернему (child) или всплывающему (popup) окну, система установит значения в ноль. Функция LoadMenu загружает указанное меню из ресурсов исполняемого файла. Параметры: идентификатор исполняемого модуля; имя или идентификатор меню. Если меню успешно загружено, то функция возвращает его дескриптор. При ошибке возвращается ноль. Теперь, чтобы привязать меню к нашему окну, необходимо указать полученный дескриптор меню в третьем с конца параметре CreateWindowEx (идентификатор меню или дочернего окна). Так как функция CreateWindowEx идет сразу же за LoadMenu, нам нет необходимости создавать отдельную переменную для хранения дескриптора меню. Мы просто вписываем eax в качестве параметра функции, потому что на этом этапе в eax должен содержаться дескриптор загруженного меню.Все остальное должно быть вам знакомо из прошлых занятий. До секции ресурсов. А вот про нее, родимую, мы поговорим подробнее. За счет использования макросов ресурсы описываются немного иначе, чем в других компиляторах. В самом начале секции ресурсов должна быть макроинструкция directory, которая определяет типы содержащихся в секции ресурсов. После нее парами следуют значения, разделенные запятыми: первое в каждой паре — идентификатор типа ресурса, а второе — имя поддиректории, содержащей ресурсы указанного типа. У нас пока что один тип ресурсов — меню, поэтому и пара лишь одна. Поддиректории размещаются ниже в этой же секции. Они объявляются макроинструкцией resource. За макроинструкцией следует имя поддиректории (соответствующее имени, указанному в макросе directory), затем тройками идут параметры ресурсов — первый параметр является идентификатором ресурса (выбирается программистом, используется для доступа к ресурсу из программы), второй параметр определяет язык, а третий — имя ресурса. Если ресурс не имеет языковой принадлежности, следует использовать константу LANG\_NEUTRAL. Для объявления различных типов ресурсов существуют специальные макроинструкции, которые должны помещаться в описании каждого ресурса.

Например, картинки bmp (битовые поля от англ. bitmap), принадлежащие к типу RT\_BITMAP, объявляются макросом bitmap. Его первый параметр — имя ресурса (соответствующее имени, указанному в макросе resource), а второй — строка, содержащая путь к файлу картинки, заключенная в кавычки. Меню, относящиеся к типу RT\_MENU, объявляются макросом menu, за которым следуют описания пунктов меню. Сам по себе макрос menu имеет лишь один параметр — имя ресурса (соответствующее имени, указанному в макросе resource). А вот макрос menuitem, описывающий пункт меню, может иметь до пяти параметров, первые два из которых обязательные, а остальные три опциональные: первый — строка, содержащая в кавычках текст пункта меню; второй — уникальный идентификатор пункта, который будет передаваться в сообщении окну, если пункт будет выбран пользователем; третий параметр (уже пошли необязательные параметры) — это один из двух возможных флагов MFR — MFR\_POPUP (всплывающее меню) и MFR\_END (последний пункт меню), каждое всплывающее меню должно завершаться, а также в конце всего меню должен быть завершающий параметр; четвертый параметр — флаг состояния пункта меню — например, MFS\_CHECKED или MFS\_DISABLED; пятый параметр — флаг типа меню MFT. Список этих и других разрешенных параметров меню вы можете найти в файле FASM\INCLUDE\EQUATES\USER32.INC в секции Menu flags. Их назначение обычно интуитивно понятно, так что, надеюсь, разберетесь самостоятельно. Добавлю только, что макроинструкция menuseparator создает разделитель и может иметь всего один параметр — MFR\_END. Раз уж мы коснулись темы ресурсов, то обязательно рассмотрите самостоятельно пример MINIPAD из папки EXAMPLES, чтобы понять, как описываются иконки и версия файла в ресурсах. В описании версии VOS\_\_WINDOWS32 означает 32-битную версию windows, VFT\_APP — тип файла "приложение"(application) или VFT\_DLL — "динамическая библиотека". Если же вам неохота каждый раз возиться с синтаксисом ресурсов, можете воспользоваться редактором ресурсов и подключать отдельный файл .res способом, указанным в начале статьи. Выбрать подходящий редактор ресурсов вам поможет страничка: [сайт](http://www.wasm.ru/toollist.php?list=2) . Теперь нам необходимо добавить в код программы обработчик сообщений от меню — без обработки от нашего меню будет мало проку. Добавим в секцию данных пару строчек:

**mb111 db 'Пункт 1-1-1',0  
mb13 db ' Пункт 1-3',0  
hmenu dd ?  
menuinfo MENUITEMINFO sizeof.MENUITEMINFO,MIIM\_STATE**  
  
Сразу после вызова функции LoadMenu сохраним дескриптор меню в переменную:  
**… … …  
invoke LoadMenu,[wc.hInstance],1  
mov [hmenu],eax  
… … …**Когда пользователь выберет пункт меню, окну-владельцу будет послано сообщение WM\_COMMAND с идентификатором пункта меню в младшем слове первого параметра (wparam) и нулем в старшем слове. Следовательно, можно считать, что весь wparam содержит идентификатор пункта меню. Хотя это и не самый оптимальный алгоритм, зато вполне наглядный — будем сравнивать wparam с нашими идентификаторами:

**proc WindowProc hwnd,wmsg,wparam,lparam  
push ebx esi edi  
cmp [wmsg],WM\_COMMAND  
je .wmcommand  
cmp [wmsg],WM\_DESTROY  
je .wmdestroy  
.defwndproc:  
invoke DefWindowProc,[hwnd],[wmsg],[wparam],[lparam]  
jmp .finish  
.wmcommand:  
cmp [wparam],111  
je .111  
cmp [wparam],13  
je .13  
cmp [wparam],21  
je .21  
jmp .finish  
  
.111:  
invoke MessageBox,0,mb111,title,MB\_OK  
jmp .finish  
.13:  
invoke MessageBox,0,mb13,title,MB\_OK  
jmp .finish  
.21:  
invoke GetMenuItemInfo,[hmenu],21,0,menuinfo  
xor [menuinfo.fState],MFS\_CHECKED  
invoke SetMenuItemInfo,[hmenu],21,0,menuinfo  
jmp .finish  
.wmdestroy:  
invoke PostQuitMessage,0  
mov eax,0  
.finish:  
pop edi esi ebx  
ret  
endp**

Структура MENUITEMINFO необходима нам здесь лишь для переключения состояния пункта меню 2-1. В первый ее элемент мы сразу помещаем размер всей структуры (sizeof.MENUITEMINFO), а во второй — маску запрашиваемых и устанавливаемых элементов структуры. MIIM\_STATE означает, что в функциях GetMenuItemInfo и SetMenuItemInfo будет использоваться элемент структуры fState, в котором содержатся флаги состояния пункта меню (MFS\_). Функция GetMenuItemInfo возвращает информацию о заданном пункте меню. Параметры: дескриптор меню; идентификатор или позиция пункта меню в зависимости от третьего параметра; если ноль, то во втором параметре содержится идентификатор пункта меню, иначе во втором параметре — его позиция; указатель на структуру MENUITEMINFO, размер структуры и маска должны быть предварительно заполнены. После вызова GetMenuItemInfo элемент fState должен содержать флаги состояния элемента меню. Обычно флаг какого-либо состояния соответствует определенному биту. Например, MFS\_CHECKED = MF\_CHECKED = 8. Число 8 в двоичном эквиваленте — это 1000b (кто не в курсе, ищите в интернете информацию о системах счисления). То есть четвертый по счету бит установлен в единицу и означает MFS\_CHECKED. Команда XOR выполняет логическую операцию исключающего ИЛИ над битами двух операндов. Если соответствующие биты операндов равны, то бит в первом операнде сбрасывается в ноль, если различны — устанавливается в единицу. По сути, биты в первом операнде переключаются в другое положение, если соответствующие биты второго операнда равны единицам, и остаются в прежнем состоянии, когда соответствующие биты второго операнда равны нулю. В нашем случае второй операнд — MFS\_CHECKED, он же 1000b в двоичном эквиваленте. Стало быть, флаг будет переключен из нуля в единицу либо из единицы в ноль. Теперь функция SetMenuItemInfo устанавливает состояние пункта меню в соответствии с заданными элементами структуры. Параметры функции аналогичны параметрам GetMenuItemInfo. При ошибке обе функции возвращают в eax ноль. Я не стану слишком подробно рассказывать о структуре MENUITEMINFO и использующих эту структуру функциях, потому что подобной информации весьма много в интернете, и, несмотря на то, что в большинстве случаев описание идет в синтаксисе С++ и Delphi, вы всегда теперь сможете провести аналогию с нашим примером.  
Все приводимые примеры были протестированы на правильность работы под Windows XP и, скорее всего, будут работать под другими версиями Windows, однако я не даю никаких гарантий их правильной работы на вашем компьютере.  
  
Исходные тексты программ вы можете найти на форуме: [сайт](http://forum.sa-sec.org/index.php?showtopic=766)  
  
**BarMentaLisk, SASecurity gr.**

*Компьютерная газета. Статья была опубликована в номере 21 за 2008 год в рубрике*[*программирование*](https://nestor.minsk.by/kg/abc/%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/)