**Как обработать нажатие Enter в edit box'е?**

**Автор: [Игорь Вартанов](mailto:ivartanov@rsdn.ru" \o "Отправить письмо" \t "_blank)**

**Опубликовано: 04.02.2002  
Исправлено: 01.09.2005  
Версия текста: 1.1**

Начнем с того, что для обработки нажатия Enter необходимо, чтобы (в общем случае) окно редактирования ожидало этого нажатия (т.е. имело стиль **ES\_MULTILINE**). В противном случае система выполнит трансляцию этого нажатия в нажатие кнопки родительского окна, имеющей в текущий момент стиль **BS\_DEFAULTPUSHBUTTON**. Кстати, это довольно неплохая методика для диалога, содержащего единственное окно ввода и имеющего кнопку по-умолчанию OK. Если же диалог (или окно) имеет несколько окон ввода, и логика работы приложения подразумевает, что нажатие Enter означает окончание ввода в выбранном окне и перевод фокуса на следующее, то скорее всего вам подойдет нижеследующая методика.

**Основной вариант**

[Демонстрационный проект EditDlg](http://www.rsdn.ru/article/qna/controls/editpret/editdlg.zip)

**WinAPI**

|  |
| --- |
| Обратите внимание, окно редактирования должно иметь стиль **ES\_MULTILINE**. |

Основная идея состоит в подмене стандартной процедуры окна редактирования (т.н. subclassing) при инициализации окна диалога, и выполнение в новой процедуре обработки нажатия клавиши. В нашем примере при обнаружении нажатия Enter выполняется копирование текста окна в буфер текста и перевод фокуса на следующий контрол диалогового окна. Если же была нажата иная клавиша, выполняется вызов стандартной оконной процедуры для окон класса "edit".

|  |
| --- |
| #include <windows.h>  #include "resource.h"  WNDPROC oldEditProc = NULL;  LRESULT CALLBACK newEditProc(HWND hEdit, UINT msg, WPARAM wParam, LPARAM lParam)  {  switch(msg)  {  case WM\_KEYDOWN:  {  if(VK\_RETURN == wParam)  {  HWND hParent = GetParent(hEdit);  SendMessage( hParent, msg, wParam, lParam);  SetFocus( GetNextDlgTabItem( hParent, hEdit, FALSE ) );  return 0; // запрет обработки по-умолчанию  }  }  break;  case WM\_CHAR:  if(VK\_RETURN == wParam)  return 0; // запрет обработки по-умолчанию  break;  }  return CallWindowProc(oldEditProc, hEdit, msg, wParam, lParam);  }  BOOL CALLBACK DlgProc(HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam)  {  static char m\_edText[256] = "";  switch(msg)  {  case WM\_INITDIALOG:  oldEditProc = (WNDPROC) SetWindowLong(  GetDlgItem(hDlg, IDC\_EDIT1),  GWL\_WNDPROC, (LONG)newEditProc);  break;  case WM\_COMMAND:  if(wParam == IDCANCEL)  EndDialog(hDlg, 0);  break;  case WM\_KEYDOWN:  if( VK\_RETURN == wParam)  GetDlgItemText(hDlg, IDC\_EDIT1, m\_edText, 256);  break;  }  return 0;  }  int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,  LPSTR lpCmdLine, int nCmdShow )  {  DialogBox(hInstance, "MAINDLG", HWND\_DESKTOP, (DLGPROC)DlgProc);  return 0;  } |

Обратите внимание на то, что обработчики сообщений при обнаружении нажатия Enter возвращают из оконной процедуры нуль. Это делается для того, чтобы сообщения не передавались обработчику по-умолчанию (и, следовательно, не выполнялось нажатие кнопки по-умолчанию).

**MFC**

|  |
| --- |
| Обратите внимание, окно редактирования должно иметь стиль **ES\_MULTILINE**. |

Для реализации поведения приложения, аналогичного только что описанному, необходимо создать класс, производный от **CEdit**, имеющий собственные обработчики сообщений **WM\_KEYDOWN** и **WM\_CHAR** (при создании класса и добавлении обработчиков используйте ClassWizard).

|  |
| --- |
| // .h-файл класса ////////////////////////////////////////////////  . . .  class CEnterEdit : public CEdit  {  public:  CEnterEdit();  public:  virtual ~CEnterEdit();  protected:  //{{AFX\_MSG(CEnterEdit)  afx\_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);  afx\_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);  //}}AFX\_MSG  DECLARE\_MESSAGE\_MAP()  };  // .cpp-файл класса //////////////////////////////////////////////  . . .  BEGIN\_MESSAGE\_MAP(CEnterEdit, CEdit)  //{{AFX\_MSG\_MAP(CEnterEdit)  ON\_WM\_KEYDOWN()  ON\_WM\_CHAR()  //}}AFX\_MSG\_MAP  END\_MESSAGE\_MAP()  void CEnterEdit::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)  {  if(nChar == VK\_RETURN)  {  // Предполагаем, что родительское окно эдит-бокса -  // диалог класса CEditDlgDlg, который имеет буфер хранения  // введенного текста m\_edText типа CString.  CEditDlgDlg\* pDlg = (CEditDlgDlg\*) GetParent();  GetWindowText(pDlg->m\_edText);  pDlg->GetNextDlgTabItem(this)->SetFocus();  return; // запрет обработки по-умолчанию  }  CEdit::OnKeyDown(nChar, nRepCnt, nFlags);  }  void CEnterEdit::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)  {  if(nChar == VK\_RETURN)  return; // запрет обработки по-умолчанию  CEdit::OnChar(nChar, nRepCnt, nFlags);  } |
| Подмена оконной процедуры - универсальный метод для получения необходимой функциональности. Если же есть возможность получить доступ к циклу сообщений, то можно воспользоваться альтернативной методикой - обработкой сообщения **WM\_KEYDOWN** в самом цикле (см. далее - Альтернативный вариант). |

Пример EditDlg демонстрирует обработку нажатия клавиши Enter. Он содержит два проекта - WinAPI и MFC.

**Альтернативный вариант**

Не всегда целесообразно обработку нажатия Enter возлагать на окно редактирования. Если в поведение приложения необходимо добавить указанную реакцию, но для самого окна достаточно обычной функциональности (однострочное окно редактирования), можно, не меняя стиля окна редактирования, самостоятельно обрабатывать нажатие Enter, анализируя содержимое сообщений в цикле обработки сообщений.

Необходимо помнить, что цикл обработки сообщений модального диалога реализуется самой системой и недоступен для программиста. В этом случае остается единственное средство - подмена оконной процедуры окна редактирования, описанная выше (см. Основной вариант).

Детали реализации этого метода очень сильно зависят от постановки задачи, среды разработки и организации цикла обработки сообщений. Общая схема такова:

1. До выполнения **DispacthMessage(&msg)** необходимо проанализировать поле msg.message на приход сообщения **WM\_KEYDOWN**.
2. Если получено сообщение **WM\_KEYDOWN**, и поле **msg.wParam** содержит **VK\_RETURN**, то выполнить вызов функции-диспетчера нажатия Enter. При этом обычно необходимо избегать передачи полученного сообщения в функцию **DispatchMessage()**, чтобы не выполнялась обработка по-умолчанию.
3. Для всех иных сообщений выполнить стандартную обработку.

**MFC**

Для программ, использующих MFC, все необходимые проверки выполняются в методе **PreTranslateMessage()** класса приложения или окна.

|  |
| --- |
| BOOL CMyWinApp::PreTranslateMessage ( MSG\* pMsg )  {  if(  ( WM\_KEYDOWN == pMsg->message ) &&  ( VK\_RETURN == pMsg->wParam )  )  {  OnEnterPressed(); // вызов диспетчера нажатия Enter  return TRUE; // запрет дальнейшей обработки  }  // стандартная обработка сообщения  return CWinApp::PreTranslateMessage ( pMsg );  } |

**WinAPI**

Для приложений WinAPI реализация цикла обработки сообщений может выглядеть таким образом:

|  |
| --- |
| . . .  while( GetMessage( &msg, NULL, 0, 0 ) )  {  if(  ( WM\_KEYDOWN == pMsg->message ) &&  ( VK\_RETURN == pMsg->wParam )  )  {  OnEnterPressed(); // вызов диспетчера нажатия Enter  continue; // запрет дальнейшей обработки  }  // стандартная обработка сообщения  TranslateMessage( &msg );  DispatchMessage ( &msg );  }  . . . |

В функции **OnEnterPressed()** вы можете анализировать, которое из окон ввода в момент нажатия имеет фокус, и в зависимости от этого принимать решение о выполнении необходимых действий, обеспечивающих логику работы приложения.

**Редкий вариант, но вдруг вам понравится...**

|  |
| --- |
| **ПРИМЕЧАНИЕ** Поскольку этот вариант является существенным только для модальных диалогов, в которых, для того чтобы добраться до цикла сообщений, необходимо применить то (сабклассинг окна диалога) или иное (постановка локального хука) ухищрение, и поскольку сказанное совершенно не относится к MFC, где модальные диалоги "от системы" практически не применяются, то мы рассмотрим только WinAPI-вариант. |

**...локальный хук?**

Условимся заранее, что теорию применения хуков вы получите из любых других источников ( например, из статьи Kyle Marsh [**Хуки в Win32**](http://www.rsdn.ru/?article/?baseserv/winhooks.xml) или Dr. Joseph M. Newcomer [**Хуки и DLL**](http://www.rsdn.ru/?article/?baseserv/hookdll.xml) на нашем сайте). Там же вы познакомитесь и с их разновидностями. Мы же продолжим решать нашу задачу - перехват нажатия Enter в модальном диалоге.

Итак, в качестве необходимого теоретического минимума заметим, что механизм "крюков" (hook - англ., крюк) позволяет приложению зарегистрировать некий обработчик, который система будет вызывать в ответ на события, происходящие в ее недрах, с целью оповещения пользовательского кода об этих событиях. Локальный хук вызывается только для событий, относящихся к процессу, поставившему хук, что практически никак не ухудшает общую производительность системы вцелом. И потому именно этот механизм подходит нам для наших целей.

Нам необходимо поставить хук типа **WH\_MSGFILTER**, который позволяет проводить мониторинг событий в диалогах (в том числе и MessageBox), меню и полосах прокрутки. Код логически распадается на относительно стандартную часть, имеющую сходное строение для хуков любого типа, и специфическую часть, которая будет выполнять для нас полезную работу. Стандартный код может выглядеть следующим образом:

|  |
| --- |
| LRESULT DlgBoxMsgFilter( UINT code, WPARAM wParam, LPARAM lParam );  HHOOK g\_hHook = NULL;  LRESULT CALLBACK HookProc( int code, WPARAM wParam, LPARAM lParam )  {  LRESULT res = 0;    // служебная обработка  if( 0 > code )  return CallNextHookEx( g\_hHook, code, wParam, lParam );  // вызов пользовательской процедуры "полезного действия"  res = DlgBoxMsgFilter( code, wParam, lParam );  if( res > -1 )  return res;  return CallNextHookEx( g\_hHook, code, wParam, lParam );  }  BOOL CALLBACK DlgProc( HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam )  {  switch( msg )  {  case WM\_INITDIALOG:  // постановка хука...  g\_hHook = SetWindowsHookEx( WH\_MSGFILTER, HookProc,  GetModuleHandle( NULL ),  GetCurrentThreadId() );  break;  case WM\_COMMAND:  switch( LOWORD(wParam) )  {  case IDCANCEL:  if( BN\_CLICKED == HIWORD(wParam) )  {  // ... и его снятие  if( g\_hHook )  UnhookWindowsHookEx( h\_hHook );  EndDialog( hDlg, 0 );  }  break;  }  break;  }  return 0;  } |

Теперь обратимся к процедуре **DlgBoxMsgFilter()**. Легко заметить, что она выполняет практически те же действия, что и **newEditProc()** из ОСНОВНОГО ВАРИАНТА, а именно - обнаружение нажатия Enter и переход на следующий контрол, имеющий стиль **WS\_TABSTOP**. Поскольку нас интересуют только события диалогов (а не меню, и не скроллбаров), то и фильтровать мы будем только коды типа **MSGF\_DIALOGBOX**.

|  |
| --- |
| LRESULT DlgBoxMsgFilter( UINT code, WPARAM wParam, LPARAM lParam )  {  LPMSG pMsg = (LPMSG)lParam;  HWND hEdit1 = GetDlgItem( g\_hDlg, IDC\_EDIT1 ),  hEdit2 = GetDlgItem( g\_hDlg, IDC\_EDIT2 );  switch( code )  {  case MSGF\_DIALOGBOX:  {  // следим за нажатиями в обоих эдитбоксах  if( hEdit1 != pMsg->hwnd && hEdit2 != pMsg->hwnd )  return -1;  switch(pMsg->message)  {  case WM\_KEYDOWN:  if( VK\_RETURN == pMsg->wParam )  {  // нажат Enter, сообщим об этом родительскому окну (диалогу)  SendMessage( g\_hDlg, pMsg->message, pMsg->wParam, pMsg->lParam );    // перейдем к следующему TABSTOP-контролу диалога  SetFocus( GetNextDlgTabItem( g\_hDlg, pMsg->hwnd, FALSE ) );  return TRUE;  }  break;  }  }  break;  }  return -1;  } |