**Работа с клавиатурой.**

Каждое окно в системе устроено таким образом, что может получать сообщения при нажатии клавиш на клавиатуре. Эти сообщения имеют два типа: виртуальные коды и символьные сообщения. Для того, чтобы преобразовывать виртуальные коды клавиш в соответствующием им символьные, к окну должен быть прикреплён цикл трансляции сообщений. Если окно отображает ввод с клавиатуры, то оно должно создавать и отображать каретку для указания места появления следующего символа.

* [Обработка нажатия клавиш](http://sources.ru/msdn/library/using_keyboard_input.shtml#processing_keystrokes)
* [Трансляция символьных сообщений](http://sources.ru/msdn/library/using_keyboard_input.shtml#translating_char_messages)
* [Обработка буквенных сообщений](http://sources.ru/msdn/library/using_keyboard_input.shtml#processing_char_messages)
* [Использование каретки](http://sources.ru/msdn/library/using_keyboard_input.shtml#using_caret)
* [Отображение ввода с клавиатуры](http://sources.ru/msdn/library/using_keyboard_input.shtml#displaying_input)

**Обработка нажатия клавиш**

При нажатии клавиш на клавиатуре в окно (а точнее в оконную процедуру), которое имеет фокус поступают сообщения [WM\_KEYDOWN](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/KeyboardInput/KeyboardInputReference/KeyboardInputMessages/WM_KEYDOWN.asp), [WM\_KEYUP](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/KeyboardInput/KeyboardInputReference/KeyboardInputMessages/WM_KEYUP.asp), [WM\_SYSKEYDOWN](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/KeyboardInput/KeyboardInputReference/KeyboardInputMessages/WM_SYSKEYDOWN.asp), и [WM\_SYSKEYUP](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/KeyboardInput/KeyboardInputReference/KeyboardInputMessages/WM_SYSKEYUP.asp). Основное сообщение, которое система отправляет окну при нажатии клавиши - **WM\_KEYDOWN**.

При поступлении сообщения **WM\_KEYDOWN**, оконная процедура должна проверить виртуальный код (который содержится в параметре *wParam* сообщения) и решить что дальше с ним делать. Обычно приложения обрабатывают только функциональные клавиши, стрелки, а так же специальные клавиши, такие как INS, DEL, HOME, и END, то есть все клавиши кроме букв и цифр.

Следующий пример демонстрирует работу обычной оконной процедуры, получающей и обрабатывающей нажатия клавиш:

case WM\_KEYDOWN:

switch (wParam)

{

case VK\_LEFT:

// Нажата стрелка влево.

break;

case VK\_RIGHT:

// Нажата стрелка враво.

break;

case VK\_UP:

// Нажата стрелка вверх.

break;

case VK\_DOWN:

// Нажата стрелка вниз.

break;

case VK\_HOME:

// Нажата клавиша HOME.

break;

case VK\_END:

// Нажата клавиша END.

break;

case VK\_INSERT:

// Нажата клавиша INS.

break;

case VK\_DELETE:

// Нажата клавиша DEL.

break;

case VK\_F2:

// Нажата клавиша F2.

break;

// Обработка других не-буквенных клавиш.

default:

break;

}

**Трансляция символьных сообщений**

Каждый поток, который получает ввод с клавиатуры, должен иметь в своём цикле сообщений функцию [TranslateMessage](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/Windowing/MessagesandMessageQueues/MessagesandMessageQueuesReference/MessagesandMessageQueuesFunctions/TranslateMessage.asp). Эта функция позволяет отделить нажатия функциональных клавиш от символьных и преобразовать виртуальный код клавиши в символьный, а так же поместить его в очередь сообщений. При этом в параметре *wParam* символьного сообщения будет содержаться код символа.

Вообще, цикл сообщений использует функцию **TranslateMessage** для трансляции всех сообщений, а не только для клавиатуры. Однако, применение **TranslateMessage** к другим типам сообщений не несёт никакого эффекта, зато гарантирует корректную трансляцию ввода с клавиатуры.

Пример:

MSG msg;

BOOL bRet;

while (( bRet = GetMessage(&msg, (HWND) NULL, 0, 0)) != 0)

{

if (bRet == -1);

{

// обработка ошибки

}

else

{

if (TranslateAccelerator(hwndMain, haccl, &msg) == 0)

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

}

**Обработка буквенных сообщений**

Если функция **TranslateMessage** перевела виртуальный код клавиши в буквенное значение, то оконная процедура получит одно из следующих сообщений: [WM\_CHAR](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/KeyboardInput/KeyboardInputReference/KeyboardInputMessages/WM_CHAR.asp), [WM\_DEADCHAR](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/KeyboardInput/KeyboardInputReference/KeyboardInputMessages/WM_DEADCHAR.asp), [WM\_SYSCHAR](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/KeyboardAccelerators/KeyboardAcceleratorReference/KeyboardAcceleratorMessages/WM_SYSCHAR.asp), и [WM\_SYSDEADCHAR](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/KeyboardInput/KeyboardInputReference/KeyboardInputMessages/WM_SYSDEADCHAR.asp). По умолчанию, оконная процедура получает только сообщение **WM\_CHAR** игнорируя при этом остальные буквенные сообщения. Функция **TranslateMessage** генерирует сообщение **WM\_CHAR** если была нажата одна из следующих клавиш:

* Любая буквенная клавиша
* BACKSPACE
* ENTER (возврат каретки)
* ESC
* SHIFT+ENTER (перевод строки)
* TAB

При поступлении сообщения **WM\_CHAR**, код символа содержится в параметре *wParam*.

Пример:

case WM\_CHAR:

switch (wParam)

{

case 0x08:

// backspace.

break;

case 0x0A:

// Перевод строки.

break;

case 0x1B:

// клавиша Esc.

break;

case 0x09:

// клавиша табуляции (tab).

break;

case 0x0D:

// возврат каретки (Enter).

break;

default:

// Обрабатываем остальные отображаемые символы.

break;

}

**Использование каретки**

Обычно, каретка (текстовый курсор) используется для отображения позиции, в которой появится следующий символ. Если окно позволяет вводить текст, то при получении фокуса, оно должно создать и отобразить каретку и наоборот, при потере фокуса, необходимо скрыть и удалить каретку. Делается это в обработчиках сообщений [WM\_SETFOCUS](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/KeyboardInput/KeyboardInputReference/KeyboardInputMessages/WM_SETFOCUS.asp) и [WM\_KILLFOCUS](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/KeyboardInput/KeyboardInputReference/KeyboardInputMessages/WM_KILLFOCUS.asp).

**Отображение ввода с клавиатуры**

Ниже представлен пример, который демонстрирует, как приложение может получать введённые символы с клавиатуры, отображать их в клиентской области окна, и обновлять положение текстового курсора с каждым введённым символом. Так же в примере показывается как перемещать каретку в ответ на следующие клавиши: СТРЕЛКА ВЛЕВО, СТРЕЛКА ВПРАВО, HOME, и END, а так же как подсветить выделенный текст в ответ на комбинацию клавиш SHIFT+СТРЕЛКА ВПРАВО.

В обработчике сообщения [WM\_CREATE](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/Windowing/Windows/WindowReference/WindowMessages/WM_CREATE.asp) оконная процедура выделяет буфер объёмом 64K для хранения ввода с клавиатуры. Так же, пример получает текущие настройки шрифта, сохраняя высоту и среднюю ширину символов. Высота и ширина используются в обработчике [WM\_SIZE](http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/Windowing/Windows/WindowReference/WindowMessages/WM_SIZE.asp), чтобы вычислить длину строки и максимальное количество строк в клиентской области.

В обработчике **WM\_SETFOCUS**, оконная процедура создаёт и отображает каретку, а в обработчике **WM\_KILLFOCUS** скрывает и удаляет её.

В обработчике **WM\_CHAR**, отображаются символы из буфера ввода, и изменение координаты каретки. Так же оконная процедура преобразует символы табуляции в четыре последовательных пробела. Символы Backspace, перевод строки и escape генерируют звуковой сигнал (beep).

В обработчике сообщения **WM\_KEYDOWN** обрабатываются перемещения каретки влево, вправо, в конец (end) и в начало (home). При обработке правой стрелки, оконная процедура проверяет состояние клавиши SHIFT и, если она нажата, то выделяет символ справа от кратки при её перемещении.

Обратите внимание, что приведённый код будет работать как в Unicode, так и в ANSI кодировках.

Пример:

#define BUFSIZE 65535

#define SHIFTED 0x8000

LONG APIENTRY MainWndProc(HWND hwndMain, UINT uMsg,

WPARAM wParam, LPARAM lParam)

{

HDC hdc; // дескриптор контекста устройства

TEXTMETRIC tm; // структура с параметрами текста

static DWORD dwCharX; // средняя ширина символов

static DWORD dwCharY; // высота символов

static DWORD dwClientX; // ширина клиентской области

static DWORD dwClientY; // Высота клиентской области

static DWORD dwLineLen; // длина строки

static DWORD dwLines; // кол-во строк в клиентской области

static int nCaretPosX = 0; // горизонтальная координата каретки

static int nCaretPosY = 0; // вертикальная координата каретки

static int nCharWidth = 0; // ширина символа

static int cch = 0; // кол-во символов в буфере

static int nCurChar = 0; // номер текущего символа

static PTCHAR pchInputBuf; // буфер ввода

int i, j; // счётчики цикла

int cCR = 0; // счётчик возвратов каретки

int nCRIndex = 0; // номер последнего возврата каретки

int nVirtKey; // код виртуальной клавиши

TCHAR szBuf[128]; // временный буфер

TCHAR ch; // текущий символ

PAINTSTRUCT ps; // требуется для BeginPaint

RECT rc; // прямоугольник для DrawText

SIZE sz; // размерность строки

COLORREF crPrevText; // предыдущий цвет текста

COLORREF crPrevBk; // предыдущий цвет фона

size\_t \* pcch;

HRESULT hResult;

switch (uMsg)

{

case WM\_CREATE:

// Получаем параметры текущего шрифта.

hdc = GetDC(hwndMain);

GetTextMetrics(hdc, &tm);

ReleaseDC(hwndMain, hdc);

// Сохраняем среднюю ширину и высоту символа.

dwCharX = tm.tmAveCharWidth;

dwCharY = tm.tmHeight;

// Выделяем буфер для хранения ввода с клавиатуры.

pchInputBuf = (LPTSTR) GlobalAlloc(GPTR,

BUFSIZE \* sizeof(TCHAR));

return 0;

case WM\_SIZE:

// Сохраняем новую ширину и высоту клиентской области.

dwClientX = LOWORD(lParam);

dwClientY = HIWORD(lParam);

// Вычисляем максимальную ширину строки и максимальное

// количество строк в клиентской области.

dwLineLen = dwClientX - dwCharX;

dwLines = dwClientY / dwCharY;

break;

case WM\_SETFOCUS:

// Когда окно получает фокус, то создаём, позиционируем

// и отображаем каретку.

CreateCaret(hwndMain, (HBITMAP) 1, 0, dwCharY);

SetCaretPos(nCaretPosX, nCaretPosY \* dwCharY);

ShowCaret(hwndMain);

break;

case WM\_KILLFOCUS:

// Когда окно теряет фокус, то скрываем и

// уничтожаем каретку.

HideCaret(hwndMain);

DestroyCaret();

break;

case WM\_CHAR:

// Проверяем, не находится ли текущее положение близко

// к концу буфера, чтобы не произошло его переполнение.

// Если так, то добавляем ноль и отображаем содержимое.

if (cch > BUFSIZE-5)

{

pchInputBuf[cch] = 0x00;

SendMessage(hwndMain, WM\_PAINT, 0, 0);

}

switch (wParam)

{

case 0x08: // backspace

case 0x0A: // перевод строки

case 0x1B: // esc

MessageBeep((UINT) -1);

return 0;

case 0x09: // tab

// Преобразуем табы в четыре последовательных пробела.

for (i = 0; i < 4; i++)

SendMessage(hwndMain, WM\_CHAR, 0x20, 0);

return 0;

case 0x0D: // возврат каретки

// Записываем возврат каретки и её позицию

// в начале новой строки.

pchInputBuf[cch++] = 0x0D;

nCaretPosX = 0;

nCaretPosY += 1;

break;

default: // отображаемые символы

ch = (TCHAR) wParam;

HideCaret(hwndMain);

// Получаем ширину сиволов и выводим

// символ.

hdc = GetDC(hwndMain);

GetCharWidth32(hdc, (UINT) wParam, (UINT) wParam,

&nCharWidth);

TextOut(hdc, nCaretPosX, nCaretPosY \* dwCharY,

&ch, 1);

ReleaseDC(hwndMain, hdc);

// Store the character in the buffer.

pchInputBuf[cch++] = ch;

// Вычисляем новую горизонтальную координат каретки.

// Если координата достигла максимума, то вставляем

// перевод каретки и перемещаем каретку

// в начало следующей строки.

nCaretPosX += nCharWidth;

if ((DWORD) nCaretPosX > dwLineLen)

{

nCaretPosX = 0;

pchInputBuf[cch++] = 0x0D;

++nCaretPosY;

}

nCurChar = cch;

ShowCaret(hwndMain);

break;

}

SetCaretPos(nCaretPosX, nCaretPosY \* dwCharY);

break;

case WM\_KEYDOWN:

switch (wParam)

{

case VK\_LEFT: // стрелка влево

// Каретка может быть перемещена только

// в начало текущей строки.

if (nCaretPosX > 0)

{

HideCaret(hwndMain);

// Получаем символ слева от каретки, вычисляем

// ширину символов, а затем вычитаем ширину

// из текущей горизонтальной координаты

// каретки, чтобы получить новую координату

ch = pchInputBuf[--nCurChar];

hdc = GetDC(hwndMain);

GetCharWidth32(hdc, ch, ch, &nCharWidth);

ReleaseDC(hwndMain, hdc);

nCaretPosX = max(nCaretPosX - nCharWidth,

0);

ShowCaret(hwndMain);

}

break;

case VK\_RIGHT: // стрелка вправо

// Перемещаем каретку вправо или, если достугнут

// перевод каретки, в начало следующей строки.

if (nCurChar < cch)

{

HideCaret(hwndMain);

// Получаем символ справа от каретки.

// Если это перевод каретки, то перемещаем

// курсор в начало следующей строки.

ch = pchInputBuf[nCurChar];

if (ch == 0x0D)

{

nCaretPosX = 0;

nCaretPosY++;

}

// Если символ не является переводом каретки,

// то проверяем, не нажата ли клавиша SHIFT.

// Если так, то инвертируем цвета текста

// и выводим символ.

else

{

hdc = GetDC(hwndMain);

nVirtKey = GetKeyState(VK\_SHIFT);

if (nVirtKey & SHIFTED)

{

crPrevText = SetTextColor(hdc,

RGB(255, 255, 255));

crPrevBk = SetBkColor(hdc,

RGB(0,0,0));

TextOut(hdc, nCaretPosX,

nCaretPosY \* dwCharY,

&ch, 1);

SetTextColor(hdc, crPrevText);

SetBkColor(hdc, crPrevBk);

}

// Получаем ширину символа и вычисляем

// новую горизонтальную координату каретки.

GetCharWidth32(hdc, ch, ch, &nCharWidth);

ReleaseDC(hwndMain, hdc);

nCaretPosX = nCaretPosX + nCharWidth;

}

nCurChar++;

ShowCaret(hwndMain);

break;

}

break;

case VK\_UP: // стрелка вверх

case VK\_DOWN: // стрелка вниз

MessageBeep((UINT) -1);

return 0;

case VK\_HOME: // HOME

// Устанавливаем каретку в верхний левый

// угол клиентской области.

nCaretPosX = nCaretPosY = 0;

nCurChar = 0;

break;

case VK\_END: // END

// Перемещаем каретку в конец текста.

for (i=0; i < cch; i++)

{

// Считаем возвраты каретки и сохраняем

// их номера.

if (pchInputBuf[i] == 0x0D)

{

cCR++;

nCRIndex = i + 1;

}

}

nCaretPosY = cCR;

// Копируем весь текст между последним возвратом

// каретки и окончанием ввода с клавиатуры

// во временный буфер.

for (i = nCRIndex, j = 0; i < cch; i++, j++)

szBuf[j] = pchInputBuf[i];

szBuf[j] = TEXT('\0');

// Устанавливаем горизонтальную координату каретки.

hdc = GetDC(hwndMain);

hResult = StringCchLength(szBuf, 128, pcch);

if (FAILED(hResult))

{

// обработчик ошибки

}

GetTextExtentPoint32(hdc, szBuf, \*pcch, &sz);

nCaretPosX = sz.cx;

ReleaseDC(hwndMain, hdc);

nCurChar = cch;

break;

default:

break;

}

SetCaretPos(nCaretPosX, nCaretPosY \* dwCharY);

break;

case WM\_PAINT:

if (cch == 0) // в буфере ввода ничего нет

break;

hdc = BeginPaint(hwndMain, &ps);

HideCaret(hwndMain);

// Устанавливаем границы прямоугольника, а затем

// рисуем в нём текст.

SetRect(&rc, 0, 0, dwLineLen, dwClientY);

DrawText(hdc, pchInputBuf, -1, &rc, DT\_LEFT);

ShowCaret(hwndMain);

EndPaint(hwndMain, &ps);

break;

// Обрабатываем другие сообщения.

case WM\_DESTROY:

PostQuitMessage(0);

// Освобождаем буфер ввода.

GlobalFree((HGLOBAL) pchInputBuf);

UnregisterHotKey(hwndMain, 0xAAAA);

break;

default:

return DefWindowProc(hwndMain, uMsg, wParam, lParam);

}

return NULL;

}