Windows services

Windows operating system implements functions of the application programming interface (API) and makes them available to application programs. The same functions are generally supported on 32-bit and 64-bit Windows. The functions are in the dynamic link libraries (dll files).

Win32 is the 32-bit application programming interface.

Developer support is available in the form of the Windows Software Development Kit (SDK):

- contains libraries to interface to a particular programming language, sample code, documentation
- freely available on the page http://msdn.microsoft.com/en-US/

The returned value is in the EAX register.

The NULL value that some parameters of the functions may take is implemented as a 32-bit constant 0.

The TRUE value is implemented as 1, FALSE as 0.

The constants are defined in the SmallWin.inc file.

Win32 API functions do not preserve EAX, EBX, ECX, and EDX!

Functions with a string parameter have two versions depending on the string encoding:

- ANSI => the name of a function is terminated by letter A
- Unicode => the name of a function is terminated by letter W

ASCII – 7-bit code

ANSI – 8-bit Microsoft-specific encoding, codes 128-255 are for special characters.

Unicode - a computing industry standard for the encoding of text expressed in most of the world's writing systems. Unicode can be implemented by different character encodings. The most commonly used encodings are UTF-8 and UTF-16.

UTF-8 uses 1 to 4 bytes encoding; the first 128 code points are the ASCII characters.

In UTF-16 the code unit is a 16-bit word, a character is encoded in one or two code units.

Example:

function MessageBox – displays a dialog box that contains a short message about the program status or error information, and a set of buttons

```
int MessageBoxA(
        HWND hwnd,
        LPCTSTR lpText,
        LPCTSTR lpCaption,
        UINT uType
);
```

Parameters

hWnd – a handle to the owner window of the message box. If this parameter is NULL, the message box has no owner window and the message box will be displayed in the middle of the screen.

lpText – a pointer to the null terminated string – the message to be displayed.

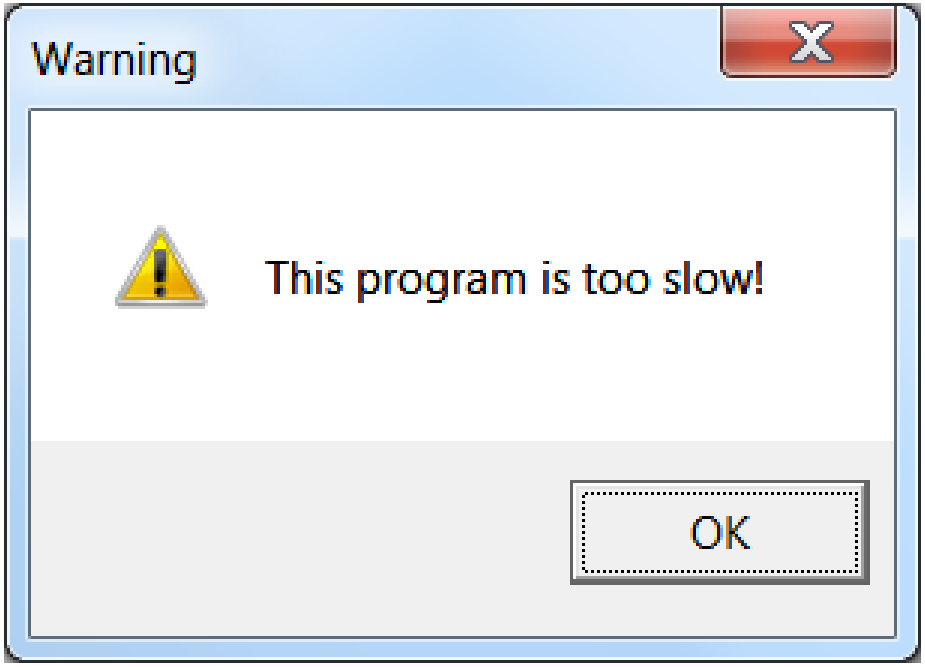
lpCaption – a pointer to the null terminated string – the dialog box title that is displayed in the box upper bar.

uType – a constant that specifies the contents and behavior of the dialog box

Return value

If the function fails, the return value is zero. Otherwise, the message box returns an integer value that indicates which button the user clicked (IDYES, IDNO, IDOK, ...).

```
int MessageBoxA(
        HWND hwnd,
        LPCTSTR lpText,
        LPCTSTR lpCaption,
        UINT uType);
```




```
.data
DialogBoxCaption DB 'Warning',0
DialogBoxText DB 'This program is too slow!',0
.code
; create message box
INVOKE MessageBoxA, NULL, offset DialogBoxText,
offset DialogBoxCaption, MB_OK or MB_ICONWARNING
```

Windows file services

The first operation with the file (e.g. create a new file, open an existing file) identifies the file by its name.

The file name is a null terminated string. It can contain the device and path specification, e.g.:

```
File1 DB 'MyFile.asm',0
File2 DB 'c:\Users\Teacher\Test.txt',0
```

The function assigns a 32-bit identification number (file handle) to the file. We have to store it because the following operations with the file (read, write, close) identify the file by this number instead of its name.

CreateFile

– creates a file and opens it for the desired type of access (read and/or write) or opens an existing file.

– returns the file handle in the EAX register. If the function fails, the return value is INVALID_HANDLE_VALUE (-1).

```
HANDLE CreateFile(
    LPCTSTR lpFileName, // address (32-bit offset) of the name
of the file
    DWORD dwDesiredAccess,
    DWORD dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD dwCreationDisposition,
    DWORD dwFlagsAndAttributes,
    HANDLE hTemplateFile
);
```

Parameter dwDesiredAccess - the requested access to the file (read, write, both, or none)

Value	Meaning
0	The application wants to query file attributes (e.g. the time of the last change) without actually accessing the file.
GENERIC_READ (80000000h)	Specifies read access to the file.
GENERIC_WRITE (40000000h)	Specifies write access to the file.
GENERIC_READ or GENERIC_WRITE	Read and write.

Parameter dwShareMode – sharing mode:

Value	Meaning
0	Prevents other processes from opening the file.
FILE_SHARE_READ (1)	Other processes can open the file for read access.
FILE_SHARE _WRITE (2)	Other processes can open the file for write access.
FILE_SHARE_READ or FILE_SHARE_WRITE	Other processes can open the file for read or write access.

Parameter lpSecurityAttributes – a pointer to a SECURITY_ATTRIBUTES structure that specifies the access rights for the file and determines whether the returned file handle can be inherited by child processes.

If this parameter is NULL, the file handle returned by CreateFile cannot be inherited and the file gets a default security descriptor (only the owner and the administrator can access to the file).

Parameter dwCreationDisposition – an action to take on a file that exists or does not exist

Value	Meaning
CREATE_NEW (1)	Creates a new file, only if it does not already exist. If the specified file exists, the function fails.
CREATE_ALWAYS (2)	Creates a new file. If the specified file exists, the function overwrites it.
OPEN_EXISTING (3)	Opens an existing file. If the specified file does not exist, the function fails.
OPEN_ALWAYS (4)	Opens a file. If the specified file does not exist, the function creates it.
TRUNCATE_EXISTING (5)	Opens an existing file and truncates it so that its size is zero bytes. If the specified file does not exist, the function fails. The file must be opened with the GENERIC_WRITE bit set in the dwDesiredAccess parameter.

Parameter dwFlagsAndAttributes
Selected attributes:

Value	Meaning
FILE_ATTRIBUTE_READONLY (1)	Applications can read the file, but cannot write to or delete it.
FILE_ATTRIBUTE_HIDDEN (2)	The file is hidden. Do not include it in an ordinary directory listing.
FILE_ATTRIBUTE_ARCHIVE (20h)	The file should be archived. Applications use this attribute to mark files for backup.
FILE_ATTRIBUTE_NORMAL (80h)	The most common value. It cannot be combined with other attributes.
FILE_ATTRIBUTE_TEMPORARY (100h)	Temporary file. The operating system tries to keep the data in memory and avoid writing the data to a hard disk (due to a faster access), because an application deletes a temporary file after a handle is closed.

Parameter hTemplateFile – a handle to a template file. The template file supplies file attributes.

When creating a new file, CreateFile ignores attributes specified in the dwFlagsAndAttributes parameter and copies the attributes of the template file.

When opening an existing file, CreateFile ignores this parameter.

This parameter can be NULL.

ReadFile

Reads data from the specified file from the position specified by the file pointer. If the function succeeds, the return value is true, otherwise false.

```
BOOL ReadFile(
HANDLE hFile, // file handle
LPVOID lpBuffer, // a pointer to the buffer that receives the data read from the file
DWORD nNumberOfBytesToRead, // the number of bytes to be read
LPDWORD lpNumberOfBytesRead, // a pointer to the variable that receives the number of bytes read
LPOVERLAPPED lpOverlapped   // a pointer to an OVELAPPED structure that is used at the asynchronous access; at the synchronous access this parameter is NULL
);
```

WriteFile

Writes data to the specified file from the position specified by the file pointer.

If the function succeeds, the return value is true, otherwise false.

The parameters are the same as with the ReadFile function.

CloseHandle

Closes the file.

```
BOOL CloseHandle(
    HANDLE hObject  // file handle
   );
```

- Write the contents of the string variable Text to a file.

```
.data
FileName          DB 'String.txt',0
Text              DB 'Hello!',0Dh,0Ah
NumberOfChar      EQU sizeof Text
FileHandle        DD ?
NumberOfBytes     DD ?

.code
main PROC
```

```
HANDLE CreateFile(
        LPCTSTR lpFileName,
        DWORD dwDesiredAccess,
        DWORD dwShareMode,
        LPSECURITY_ATTRIBUTES lpSecurityAttributes,
        DWORD dwCreationDisposition,
        DWORD dwFlagsAndAttributes,
        HANDLE hTemplateFile);
```

```
; crate file FileName
      INVOKE CreateFileA, offset FileName,
      GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
      FILE_ATTRIBUTE_NORMAL, NULL
      mov FileHandle,eax; store file handle
```

```
BOOL  WriteFile(
        HANDLE hFile,
        LPVOID lpBuffer,
        DWORD nNumberOfBytesToWrite,
        LPDWORD lpNumberOfBytesWritten,
        LPOVERLAPPED lpOverlapped );
```

```
        ; write the variable Text to the file
        INVOKE WriteFile, FileHandle, offset Text,
        NumberOfChar, offset NumberOfBytes, NULL

        ; close the file
        INVOKE CloseHandle, FileHandle

        exit
main ENDP
```

- Read an unsigned integer as a string from the text file Number.txt. Calculate its value and store it to the AX register.

File Number.txt contains:

```
4096↵
```

```
 34 30 39 36 0D 0A
```

```
.data
FileName          DB 'Number.txt',0
Char              DB ?
NumberOfChar      EQU 1
FileHandle        DD ?
NumberOfBytes     DD ?
Ten               DW 10

.code
main PROC
```

```
HANDLE CreateFile(
        LPCTSTR lpFileName,
        DWORD dwDesiredAccess,
        DWORD dwShareMode,
        LPSECURITY_ATTRIBUTES lpSecurityAttributes,
        DWORD dwCreationDisposition,
        DWORD dwFlagsAndAttributes,
        HANDLE hTemplateFile);
```

```
; open the file for read access
      INVOKE CreateFileA, offset FileName,
      GENERIC_READ, 0, NULL, OPEN_EXISTING,
      FILE_ATTRIBUTE_NORMAL, NULL
      mov FileHandle,eax; store file handle
```

```
BOOL  ReadFile(
HANDLE hFile,
LPVOID lpBuffer,
DWORD nNumberOfBytesToRead,
LPDWORD lpNumberOfBytesRead,
LPOVERLAPPED lpOverlapped );
```

```
        xor eax,eax
; read by one digit and convert to a value
Read:
        push eax ; save temporary result
        INVOKE ReadFile, FileHandle, offset Char,
        NumberOfChar, offset NumberOfBytes, NULL
        pop eax ; restore temporary result
```

```
        cmp Char,0Dh; Enter?
        je Finish
        mul Ten; ax = ax*10
        movzx cx,Char
        sub cl,'0'
        add ax,cx
        jmp Read

Finish:
        ; close the file
        INVOKE CloseHandle, FileHandle
        exit
main ENDP
```

- Write the signed integer in the AX register to the text file Number.txt.

```
.data
FileName          DB 'Number.txt',0
String            DB 5 dup(?)
Minus             DB '-'
NumberOfChar      EQU 1
FileHandle        DD ?
NumberOfBytes     DD ?
Ten               DW 10
.code
main PROC
```

```
HANDLE CreateFile(
        LPCTSTR lpFileName,
        DWORD dwDesiredAccess,
        DWORD dwShareMode,
        LPSECURITY_ATTRIBUTES lpSecurityAttributes,
        DWORD dwCreationDisposition,
        DWORD dwFlagsAndAttributes,
        HANDLE hTemplateFile);
```

```
; open the file for write access
      INVOKE CreateFileA, offset FileName,
      GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
      FILE_ATTRIBUTE_NORMAL, NULL
      mov FileHandle,eax; store file handle
```

```
BOOL  WriteFile(
HANDLE hFile,
LPVOID lpBuffer,
DWORD nNumberOfBytesToWrite,
LPDWORD lpNumberOfBytesWritten,
LPOVERLAPPED lpOverlapped );
```

```
      mov ax,-1234
      mov edi,0
      cmp ax,0
      jge Convert
; write "-" to the file and negate ax
      push eax
      INVOKE WriteFile, FileHandle, offset Minus,
NumberOfChar, offset NumberOfBytes, NULL
      pop eax
      neg ax
```

```
; convert the number in ax to a string
Convert:
  mov dx,0
  div Ten
  add dl,'0'
  mov String[edi],dl; store the remainder (one digit)
  inc edi
  cmp ax,0; stop division?
  jne Convert
```

```
; write String to the file in the reverse order
Write:
      dec edi
      mov ebx,offset String
      add ebx,edi; ebx points to the current digit
      INVOKE WriteFile, FileHandle, ebx,
NumberOfChar, offset NumberOfBytes, NULL
      cmp edi,0
      jne Write

; close the file
      INVOKE CloseHandle, FileHandle
      exit
main ENDP
```

MS-Windows programming

When a Windows application starts, it creates either a console window or a graphical window.

Create the console window in the Visual Studio:

Project – Properties – Configuration Properties – Linker – System

SubSystem: Console (/SUBSYSTEM: CONSOLE)

Output to the console window

- by calling Windows API functions.

Windows function WriteConsole writes a character string beginning at the current cursor location.

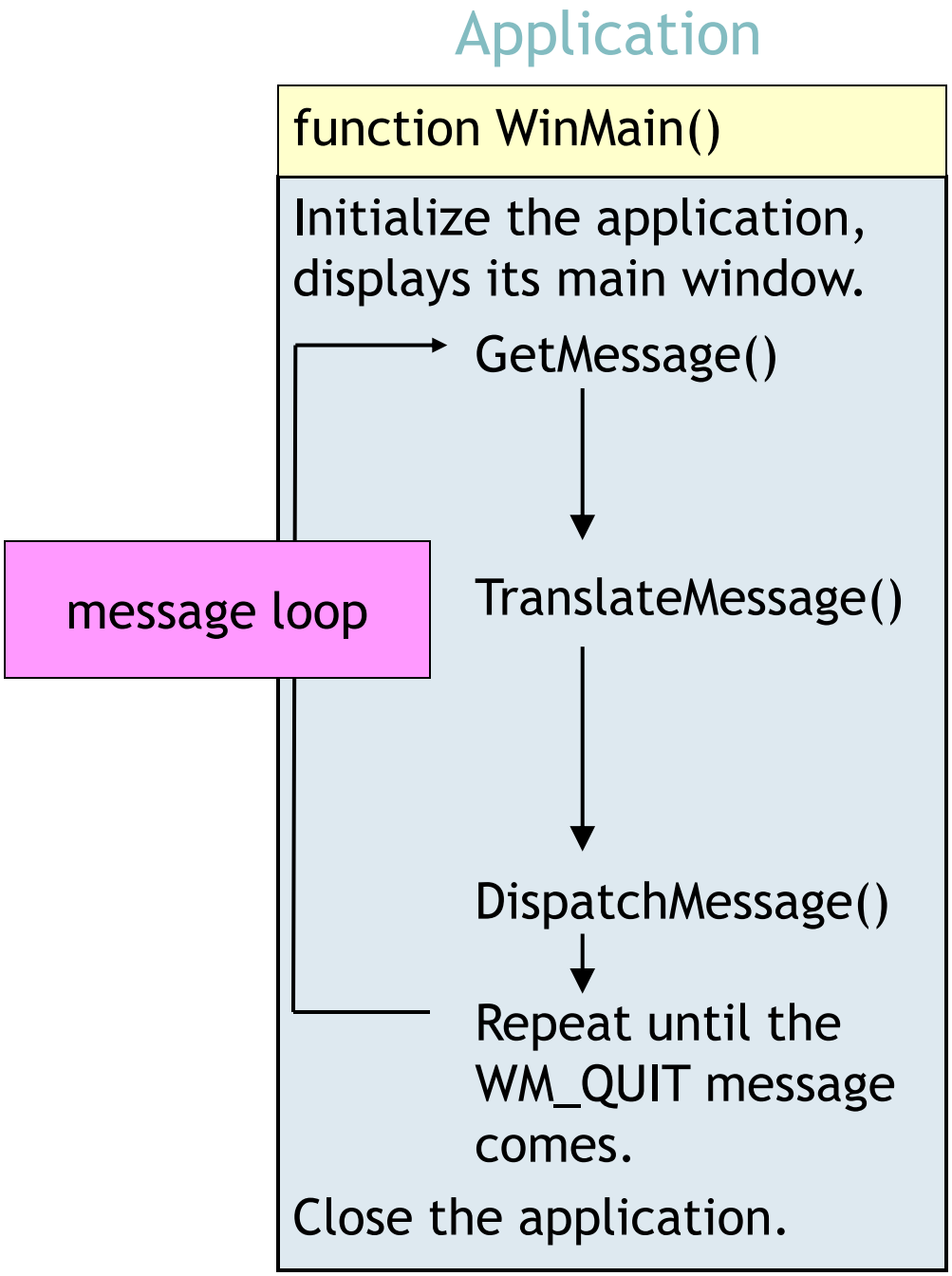
Procedure WriteString from Irvine32 library is actually a wrapper around a more detailed call to the Win32 WriteConsole function.

Input in Windows

Windows-based applications are event-driven. They do not make explicit function calls to obtain input. Instead, they wait for the system to pass input to them.

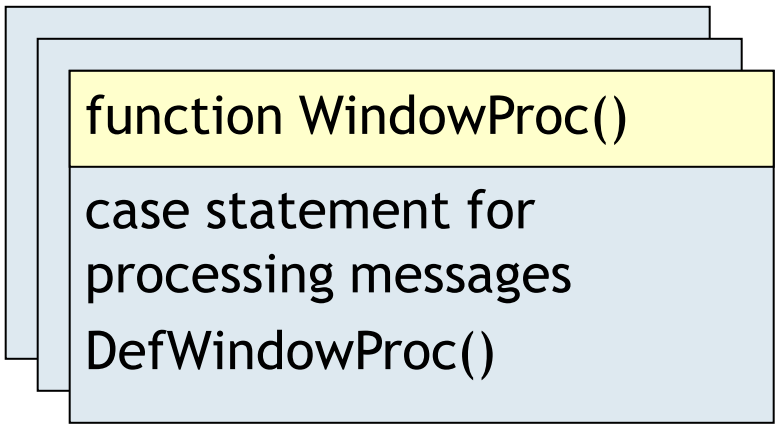
Input devices: keyboard, mouse.

Moving the mouse, click, pressing a key are input events. Windows sends messages about the input events to the application program.


Application
Windows
function WinMain()
Initialize the application, displays its main window.
GetMessage()
TranslateMessage()
DispatchMessage()
Repeat until the WM_QUIT message comes.
Close the application.
message loop
Retrieves a message from the queue and copies it to a structure of type MSG.
Translates the virtual-key message into a character message and places it back into the application message queue.
Sends a message to the window specified in the MSG structure by calling its window procedure.

Message

```
typedef struct tagMSG {
    HWND hwnd; // handle to the window that is to receive the message
    UINT message; // message identifier
    WPARAM wParam; // additional information about the message
    LPARAM lParam; // additional information
    DWORD time; // the time at which the message was posted
    POINT  pt;  // the cursor position, in screen coordinates, when the message was posted
} MSG;
```




```
LRESULT WindowProc(
  HWND hwnd,   // handle to the window
  UINT uMsg,   // message identifier
  WPARAM wParam, // additional info
  LPARAM lParam
);
```

Each application's window has a window procedure. A window procedure is a function that receives and processes all messages sent to the window. The system sends a message to a window procedure by passing the message data as arguments to the procedure.

WindowProc in the case statement checks the message identifier uMsg and performs an appropriate action; while processing the message, uses information specified by the wParam and lParam parameters.

If the window procedure ignores a message, it must send the message back to the system for default processing by calling the DefWindowProc function.

Console input

A console formats each input event (such as a single keystroke, a movement of the mouse, or a mouse-button click) as an input record that it places in the console's input buffer.

Applications can access a console's input buffer indirectly by using the high-level console I/O functions, or directly by using the low-level functions. The high-level input functions filter and process the data in the input buffer, returning only a stream of input characters.

Windows function ReadConsole reads keyboard input from a console's input buffer and returns the characters read.

Procedure ReadString from Irvine32 library wrapes the Win32 ReadConsole function.