Programming in Graphical Environment

Windows API Lecture 1

Paweł Aszklar
pawel.aszklar@pw.edu.pl

Faculty of Mathematics and Information Science
Warsaw Univeristy of Technology

Warsaw 2024

Windows API

Application Programming Interface used by all programs running on Windows family OSs
Exposed as:

- **C API** (usually labelled Win32 API)
- *COM interfaces* (sometimes included under Win32 label)
- Windows Runtime Objects

Implementation opaque to the user, execution:

- In-process (user- or kernel-mode)
- Out-of-process (local or remote)
- Changes between versions

Windows API

Provided functionality:

- memory, file system, network, devices, processes, threads management
- **user interface, user input, common UI elements**
- *shell integration*, access to windows services and registry, IPC, RPC
- system administration, application installation, diagnostics
- audio, video, multimedia, 3D and **2D graphics**
- and more…

Win32 is not Windows NT native API, however, native API:

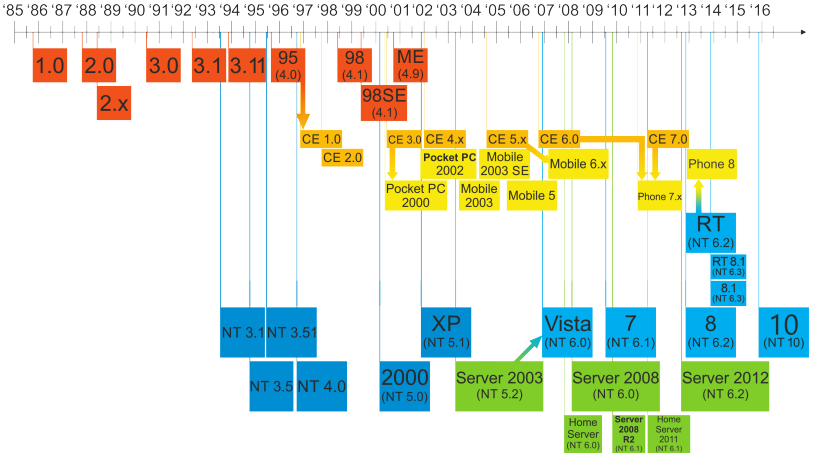
- mostly undocumented
- tends to change between versions
- not that useful without Win32 abstraction

Notes on Windows Runtime API

- Object oriented ABI (*application binary interface*)
- Language Projections: generated API, uses native language features
- Available for: C++ (C++/WinRT), C#, Visual Basic, JavaScript,…
- *COM done right*
- Exposes subset of Windows API to UWP apps (subject to their restrictions: app contenerisation, fine grained permissions, …)
- Partially built on top of Win32 API, although some features WinRT exclusive
- Significant portion usable outside of UWP apps (notable exception: XAML-based UI — requires XAML islands, Windows 10 build 17709+)
- Windows 8+ only

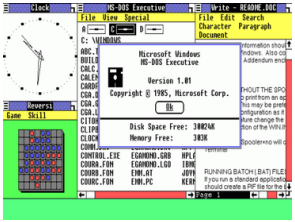
Windows OS History

- Designed in 16-bit era (Win16)
- Ported to 32-bit in NT 4.0 preserving much compatibility
- XP — first Windows OS with 64-bit version
- 32-bit and 64-bit programs using WinAPI can share codebase

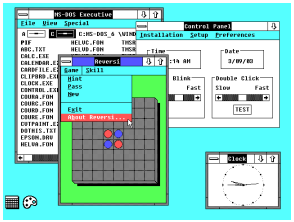


Windows™ OS Versions

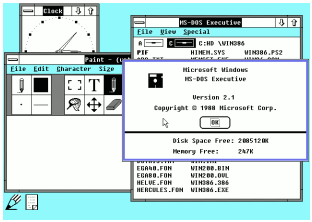
- 1.0 — 1985



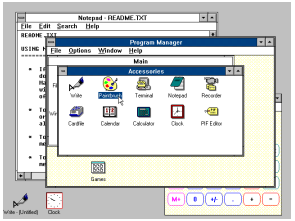
- 2.0 — 1987



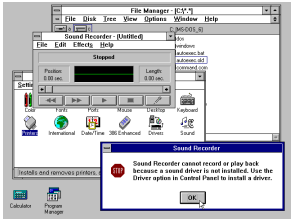
- 2.1 — 1988



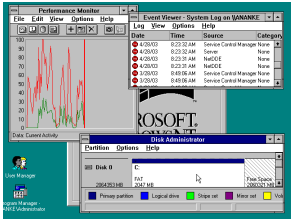
- 3.0 — 1990



- 3.1 — 1992



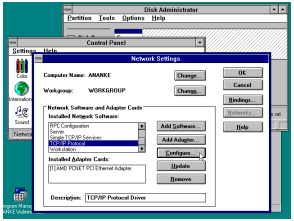
- NT 3.1 — 1993



Screenshots: © Microsoft Corporation. Used with permission from Microsoft.

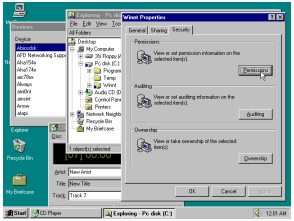
Windows™ OS Versions

- NT 3.5 — 1994

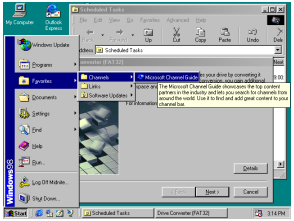


- 95 — 1995

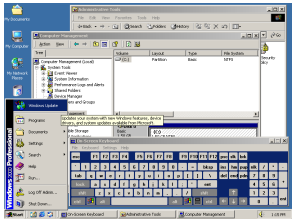
- NT 4.0 — 1996



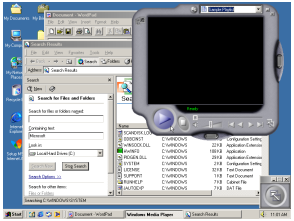
- 98 — 1998



- 2000 (NT 5.0) — 2000



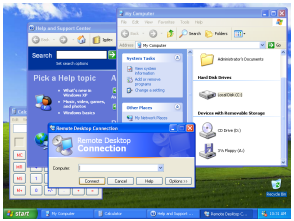
- ME — 2000



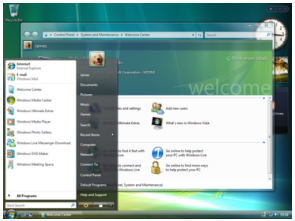
Screenshots: © Microsoft Corporation. Used with permission from Microsoft.

Windows™ OS Versions

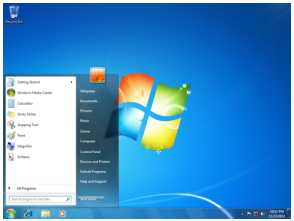
- XP (NT 5.1) — 2001



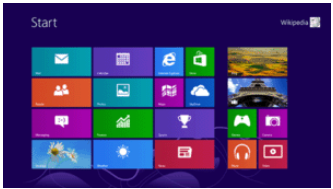
- Vista (NT 6.0) — 2007



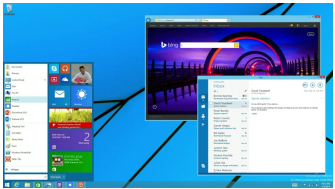
- 7 (NT 6.1) — 2009



- 8 (NT 6.2) — 2012



- 8.1 (NT 6.3) – 2013



- 10 (NT 10.0) — 2015

Recommended Reading I

Windows Dev Center

Programming Reference for Windows API

https://learn.microsoft.com/pl-pl/windows/win32/desktop-app-technologies

https://learn.microsoft.com/pl-pl/windows/win32/api/

Charles Petzold

Programming Windows, Fifth Edition

Microsoft Press, 1998

The book to read as an intro to Windows GUI programming

Feng Yuan

Windows Graphics Programming: Win32 GDI and DirectDraw

Prentice Hall Professional, 2001

Book focused on 2D drawing using GDI

Recommended Reading II

Jeffrey Richter, Christophe Nassare

Windows via C/C++, Fifth Edition

Microsoft Press, 2008

Advanced system programming topics

P. Yosifovich, A. Ionescu, M.E. Russinovich, D.A. Salomon

Windows Internals Seventh Edition, Part 1

Microsoft Press, 2017

Overview of design and inner workings of Windows NT architecture

Raymond Chen

The Old New Thing Blog

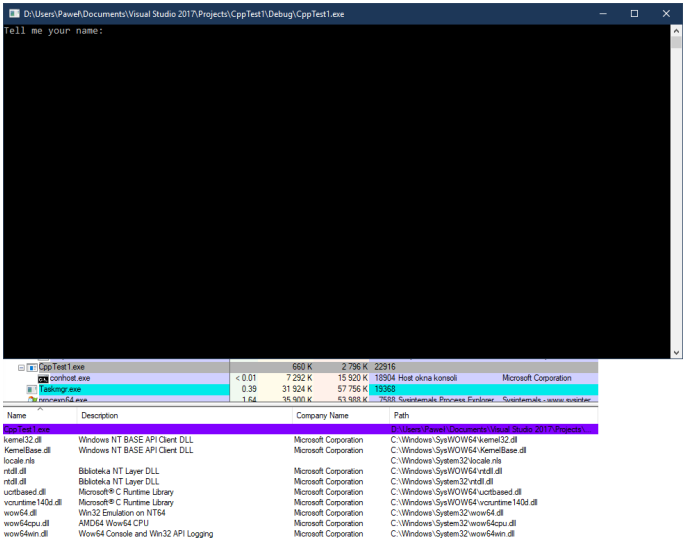
`https://devblogs.microsoft.com/oldnewthing/`

Table of Contents: `https://github.com/mity/mctrl/wiki/Old-New-Win32API`

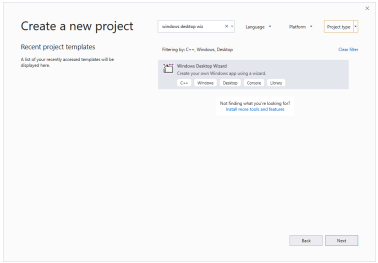
In-depth discussion on historical Windows API design choices and various edge cases.

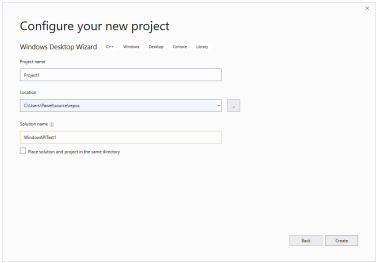
First WinAPI Program

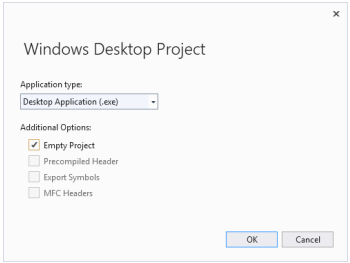
```
#include <stdio.h>
#include <string.h>

int main()
{
    char name[100] = "";
    puts("Tell me your name:");
    fgets(name, 100, stdin);
    auto len = strlen(name);
    if(len)
        name[len-1] = '\0';
    printf("Hello %s!", name);
}
```



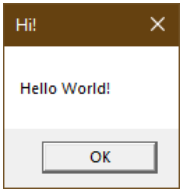
First WinAPI program — For Real This Time








```
#include <windows.h>

int APIENTRY wWinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                      LPWSTR lpCmdLine, int nShowCmd)
{
    MessageBoxW(nullptr, L"Hello World!", L"Hi!", MB_OK);
    return 0;
}
```



Data Types

Type aliases used extensively:

Win32	x86	x86_64	Win32	x86	x86_64
BOOL	int		BOOLEAN	unsigned char	
BYTE	unsigned char		CHAR	char	
WORD	unsigned short		SHORT	short	
DWORD32	unsigned int		INT	int	
DWORD	unsigned long		LONG	long	
QWORD	unsigned __int64		LONG64	__int64	
SIZE_T	unsigned long	unsigned __int64	SSIZE_T	long	__int64
ULONG_PTR	unsigned long	unsigned __int64	LONG_PTR	long	__int64
WPARAM	unsigned long	unsigned __int64	LPARAM	long	__int64
WCHAR	wchar_t		FLOAT	float	
LPVOID	const void *		HANDLE	const void *	
LPSTR	char *		LPWSTR	wchar_t *	

Data Types

- Various *TYPE*_PTR/U*TYPE*_PTR exist, mapped to `signed`/`unsigned` variants of `long` (x86), `__int64` (x86_64)

Pointer types:

- P*TYPE* or LP*TYPE* — pointer to *TYPE*

  Note: on 16-bit had *near* and *far*/*long* pointers
- PC*TYPE*, LPC*TYPE* — pointer to `const` *TYPE*
- `STR`, `WSTR` suffix — null-terminated (byte or wide) character string
- `H` prefix — various handles

  Note: maps to `HANDLE` on x86 and x86_64

  Note: handles are opaque, can contain an offset, index, an address in address space of current process, kernel or a different process. Do not try to dereference them!

Usage of WinAPI type aliases recommended for GUI code:

- Allow for code independent of platform and architecture
- Windows can run on x86 (32-bit/IA-32, 64-bit/x86_64), ARM, ARM64 architectures
- Used to run on 16-bit x86, DEC Alpha, MIPS, PowerPC, Itanium (IA-64)

Hungarian Notation

Parameters, `struct` fields often have prefixes indicating type/meaning:

Prefix	Meaning	Prefix	Meaning
`b`	boolean or byte	`c`	count
`ch`	character `CHAR`, `WCHAR`	`dw`	`DWORD`
`f`	flag(s) (boolean or bitwise)	`fn`	function
`h`	handle	`i`	`INT` or index
`l`	`LONG` or `LPARAM`, `DWORD`	`lp`	long pointer
`n`	integer `INT` or `SHORT`	`p`	pointer
`pv`	pointer to `void`	`rg`	range (array)
`sz`	null-terminated string (array or pointer to `CHAR`/`WCHAR`)	`u`	`unsigned` type (default `UINT`)
`w`	`WORD` or `WPARAM`, `UINT`	`x`,`y`	coordinates
`vk`	virtual key code	`pt`	point (`LONG` coords, `POINT`)
`pts`	point (`SHORT` coords, `POINTS`)	`rc`	rectangle (`RECT`)

Hungarian Notation

Prefixes can be combined:

`lpsz`	long pointer to null-terminated string
`cx`,`cy`	count along *X*,*Y*, i.e. width, height
`cb`,`cch`	byte, character count
`us`,`ui`,`ul`,`ull`	`USHORT`, `UINT`, `ULONG`, `ULONGLONG`

- Vestigial, no powerful IDEs in the past
- Use in your own code discouraged.

Character Encoding

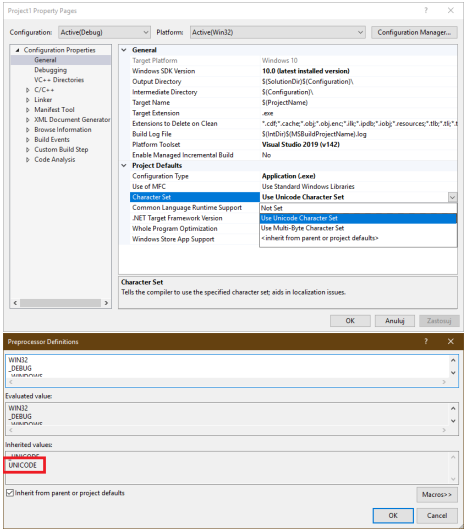
WinAPI parts dealing with characters come in two flavours:

- *"Unicode"* 16-bit wide UCS-2 characters (support for UTF-16 limited)
  - Entry point: `wWinMain`
  - Type names: `WCHAR`, `LPWSTR`, `LPCWSTR`, etc.
  - `W` function names suffix: `MessageBoxW`, `RegisterClassExW`, `CreateWindowExW`, etc.
  - Wide string literals: `L"Hello World!"`
- Single- or multi-byte (variable length encoding) *"ANSI"* characters (uses user selected codepage, UTF-8 codepage currently in beta)
  - Entry point: `WinMain`
  - Type names: `CHAR`, `LPSTR`, `LPCSTR`, etc.
  - `A` function names suffix: `MessageBoxA`, `RegisterClassExA`, `CreateWindowExA`, etc.
  - String literals: `"Hello World!"` (must match user codepage!)

Character Encoding

- Alternative approach — variant selected via preprocessor:
  - Entry point: `_tWinMain`
  - Type names: `TCHAR`, `LPTSTR`, `LPCTSTR`, etc.
  - No function names suffix: `MessageBox`, `RegisterClassEx`, `CreateWindowEx`
  - String literals: `TEXT("Hello World!")`

"Unicode" variant selected if `UNICODE` symbolic constant is defined. Otherwise *"ANSI"* variant is used



Character Encoding

- *"Unicode"* flavour used internally by Windows
- It is encouraged to use it in new programs
- `UNICODE` should be defined even if wide variant used explicitly
- Lecture slides will use it exclusively, if only to show where variants differ
- Pick one of the three approaches and use it consistently
- Mixing them, while possible, leads to strange results

Application Entry Point — (w)WinMain

```
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPWSTR lpCmdLine, int nShowCmd);
```

- `WINAPI` — calling convention
- `hInstance` — application instance handle what's that?
- `hPrevInstance` — always `nullptr` why?
- `lpCmdLine` — command line arguments (w/o program name, not split like w/ `argv`, `argc`)
- `nShowCmd` — how main window should be shown (we'll get to that)

Typical Application Lifetime

Simplest case: application with a single main window.

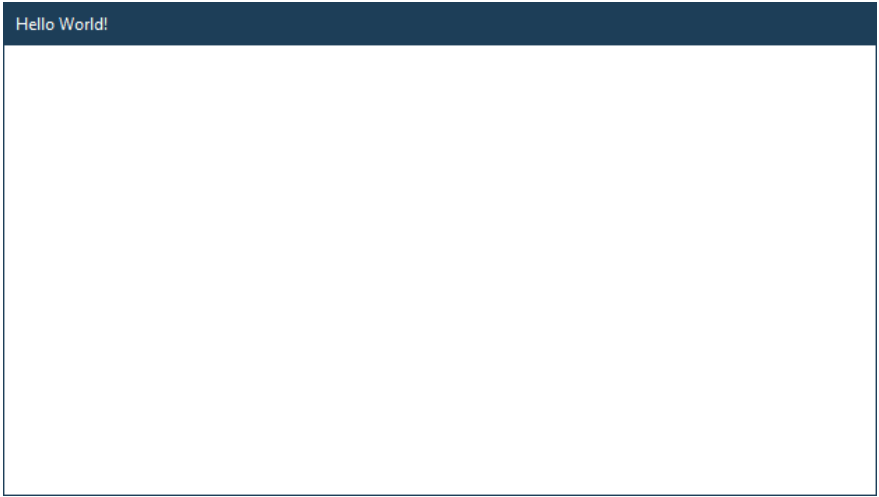
1. Register main window class
2. Create and show main window
3. Repeat
    1. Wait for messages (events)
    2. If window closed, go to 4.
    3. Respond to messages
4. Destroy main window and exit

Step 3. — the message loop — is where most of program's work is done Note: Messages and message loops will be discussed later.

Minimal (Somewhat) Working Example

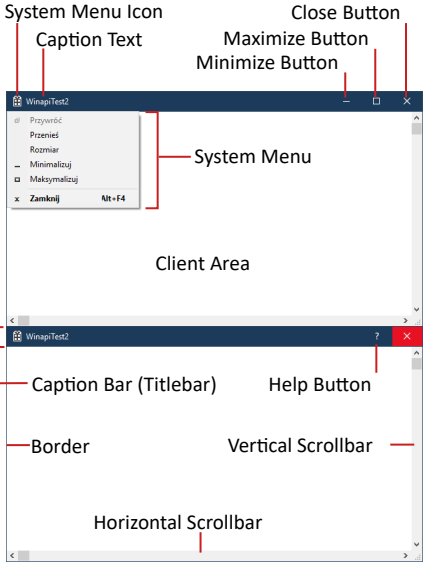
```
//disable rarely used APIs for faster builds
#define WIN32_LEAN_AND_MEAN
//disable min and max macros
//that conflict with standard C/C++
#define NOMINMAX
#include <windows.h>

int WINAPI wWinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPWSTR lpCmdLine, int nShowCmd)
{
    //Register Class
    LPCWSTR clsname = L"My Window Class";
    WNDCLASSEXW wcx{
        .cbSize = sizeof(WNDCLASSEXW)
        .lpfnWndProc = DefWindowProcW;
        .lpszClassName = clsname;
    };
    ...
    ...
    ATOM clsid = RegisterClassExW(&wcx);
    if (!clsid) return -1;
    //Create and Show Window
    HWND hWnd = CreateWindowExW(0,
        clsname/*or clsid*/, L"Hello World!",
        0, 100, 100, 640, 360, nullptr,
        nullptr, nullptr, nullptr);
    if (!hWnd) return -1;
    ShowWindow(hWnd, nShowCmd);
    //Process Messages
    MSG msg{};
    while (GetMessageW(&msg,
                nullptr, 0, 0) > 0) {
        DispatchMessageW(&msg);
    }
    //Destroy Window and Exit (Unreachable!)
    DestroyWindow(hWnd);
    return 0;
}
```

Minimal (Somewhat) Working Example



Window Anatomy

- All elements (except client area) optional, controlled by class and window styles.
- No caption bar doesn't disable system menu (can be accessed from taskbar)
- Disabling system menu removes icon and buttons
- Minimize, maximize, close options can be disabled individually (buttons greyed out if present)
- Disabling both maximize and minimize buttons removes them from caption bar
- Help button can only appear if maximize and minimize buttons are not present
- Different border styles with varying behaviour
- Border is always present if caption bar is enabled



Registering Window Class

Functions: `RegisterClassW`, `RegisterClassExW`

Parameters: `WNDCLASSW`, `WNDCLASSEXW`

- `cbSize` Size in bytes, common pattern, allows WinAPI to distinguish `struct` versions
- `style` Class style and behaviour, see: docs ▸ Appendix
- `lpfnWndProc` Callback, processes window messages
- `cbWndExtra` Additional memory internally allocated for window data for application specific use (`SetWindowLongPtrW`, `GetWindowLongPtrW`, etc.)
- `hInstance` Application/module instance, differentiates classes registered by `.exe`, `.dlls`

```
ATOM RegisterMyClass(
    HINSTANCE hInst, LPCWSTR clsn)
{
    WNDCLASSEXW wcx{};
    wcx.cbSize = sizeof(wcx);
    wcx.style =
        CS_VREDRAW | CS_HREDRAW;
    wcx.lpfnWndProc = MyWndProc;
    wcx.cbWndExtra = 0;
    wcx.hInstance = hInst;
    wcx.hIcon = nullptr;
    wcx.hCursor =
    LoadCursorW(nullptr,
        IDC_ARROW);
    wcx.hbrBackground =
        reinterpret_cast<HBRUSH>(
            COLOR_WINDOW + 1);
    wcx.lpszMenuName = nullptr;
    wcx.lpszClassName = clsn;
    wcx.hIconSm = nullptr;
    return RegisterClassExW(&wcx);
}
```

Registering Window Class

Functions: `RegisterClassW`, `RegisterClassExW`

Parameters: `WNDCLASSW`, `WNDCLASSEXW`

`hIcon` Icon handle for taskbar,  . If `nullptr` system default used.

`hCursor` Cursor for window's client area

`hbrBackground` Handle for brush used to erase window's client area

`lpszMenuName` Name of main menu resource

`lpszClassName` Name of the window class, used (with `hInstance`) to identify it

`hIconSm` System menu icon handle (for caption/title bar). If `nullptr` `hIcon` is used.

Note: resources (menus, icons, cursors, brushes) will be discussed later

```
ATOM RegisterMyClass(
    HINSTANCE hInst, LPCWSTR clsn)
{
    WNDCLASSEXW wcx{};
    wcx.cbSize = sizeof(wcx);
    wcx.style =
        CS_VREDRAW | CS_HREDRAW;
    wcx.lpfnWndProc = MyWndProc;
    wcx.cbWndExtra = 0;
    wcx.hInstance = hInst;
    wcx.hIcon = nullptr;
    wcx.hCursor =
    LoadCursorW(nullptr,
        IDC_ARROW);
    wcx.hbrBackground =
        reinterpret_cast<HBRUSH>(
            COLOR_WINDOW + 1);
    wcx.lpszMenuName = nullptr;
    wcx.lpszClassName = clsn;
    wcx.hIconSm = nullptr;
    return RegisterClassExW(&wcx);
}
```

Class Styles

Some useful class styles:

`CS_DBLCLKS`	Window will receive mouse double-click messages.
`CS_DROPSHADOW`	Enables drop-shadow effect imitating window hovering over other content. Doesn't work well on windows with caption bar.
`CS_HREDRAW`	Invalidates (causes redraw) of the whole window on width changes. Otherwise only the new part is invalidated.
`CS_VREDRAW`	As above, but for height changes.
`CS_NOCLOSE`	Disables close button and Close system menu option.

Creating a Window

Functions: `CreateWindowW`, `CreateWindowExW`

`dwExStyles` Window extended styles and behaviour, see: docs ▸ Appendix

`lpClassName` Class name or its `ATOM` id

`lpWindowName` Window name/caption text. Caption-less windows can use it for different purpose

`dwStyle` Window styles and behaviour, see: docs ▸ Appendix

`x` x-coordinate of upper-left window corner

`y` y-coordinate of upper-left window corner$^{(*)}$

`nWidth` Height of the window

`nHeight` Width of the window

Note: Window sizing and positioning will be discussed later

```
HWND CreateWindowExW(
    DWORD      dwExStyle,
    LPCWSTR    lpClassName,
    LPCWSTR    lpWindowName,
    DWORD      dwStyle,
    int        X,
    int        Y,
    int        nWidth,
    int        nHeight,
    HWND       hWndParent,
    HMENU      hMenu,
    HINSTANCE  hInstance,
    LPVOID     lpParam
);
```

Creating a Window

Functions: `CreateWindowW`, `CreateWindowExW`

- `hMenu` Menu handle (class menu used if `nullptr`) or application-determined child window identifier (only for child windows)
- `hWndParent` Parent (for child window) or owner (for overlapped or pop-up window)
- `hInstance` Application/module registering the class
- `lpParam` Custom data sent with `WM_CREATE` message

Note: Resources, messages, relations between windows will be discussed later

```
HWND CreateWindowExW(
    DWORD      dwExStyle,
    LPCWSTR    lpClassName,
    LPCWSTR    lpWindowName,
    DWORD      dwStyle,
    int        X,
    int        Y,
    int        nWidth,
    int        nHeight,
    HWND       hWndParent,
    HMENU      hMenu,
    HINSTANCE  hInstance,
    LPVOID     lpParam
);
```

Window Styles

Some useful window styles:

`WS_OVERLAPPED`	Default. Overlapped top-level window with caption bar (caption text only) and thin border.
`WS_POPUP`	Top-level pop-up window. Disables caption and border. Mutually exclusive w/ `WS_CHILD`.
`WS_CHILD`	Child window. Disables caption and border. Mutually exclusive w/ `WS_POPUP`.
`WS_BORDER`	Re-enables thin border.
`WS_CAPTION`	Re-enables caption bar.
`WS_SYSMENU`	Enables system menu, caption icon, close button.
`WS_MINIMIZEBOX`	Enables minimize button/system menu option. No effect w/o `WS_SYSMENU`.
`WS_MAXIMIZEBOX`	Enables maximize button/system menu option. No effect w/o `WS_SYSMENU`.

Window Styles

Some useful window styles:

`WS_SIZEBOX`	Enables sizing border (even w/o `WS_BORDER`).
`WS_VISIBLE`	Window becomes visible immediately (no need to call `ShowWindow`).
`WS_DISABLE`	Window starts disable (doesn't receive input messages). To re-enable it call `EnableWindow`.
`WS_MINIMIZE`	Window starts minimized (no need for `WS_MINIMIZEBOX`, YMMV).
`WS_MAXIMIZE`	Window starts maximized (no need for `WS_MAXIMIZEBOX`).
`WS_DLBFRAME`	Enables dialog-like frame. Window cannot have caption bar.
`WS_OVERLAPPEDWINDOW`	Overlapped Top-level window. Includes caption bar, system menu, minimize and maximize buttons, sizing border
`WS_POPUPWINDOW`	Top-level pop-up window. Includes border and system menu.

Extended Window Styles

Some useful extended window styles:

`WS_EX_CLIENTEDGE`	Adds a sunken inner edge around client area
`WS_EX_DLGMODALFRAME`	Adds a raised outer edge
`WS_EX_STATICEDGE`	Adds a slightly sunken outer edge (usually indicating control not accepting input)
`WS_EX_TOOLWINDOW`	Makes caption bar, close button smaller. Window by default will not appear on taskbar.
`WS_EX_CONTEXTHELP`	Adds a help button (needs caption bar, system menu, and neither minimize nor maximize buttons)
`WS_EX_NOACTIVATE`	Window can't be activated by clicking on it.
`WS_EX_APPWINDOW`	Forces a top-level window onto taskbar (if not already there, e.g. owned windows, `WS_EX_NOACTIVATE`, `WS_EX_TOOLWINDOW`)
`WS_EX_TOPMOST`	Windows stays on top of other non-topmost windows even when not active.

Window Destruction

To destroy a window call `DestroyWindow`

- Hides the window, owned windows
- Destroys child/owned windows
- Frees window resources (timers, menus, etc.)
- Invalidates window handle (`hWnd` must not be used afterwards)

Each window program creates should be destroyed once no longer needed!

End of Windows API Lecture 1

Thank you for listening! ☺

Window Frame Visual Appearance and Behaviour Guide

- Visual appearance and behaviour of window frame (a.k.a. the non-client area) controlled with styles
- Flags set as class styles, window styles, extended window styles
- Large number of flag combinations possible
- Some flags change their effect in combination with other
- Some effects are mutually exclusive
- Extensive tests needed to determine style combinations producing certain effects
- Next few slides present a comprehensive styling guide as tested on Windows 10 ver. 1809 (build 10.0.17763.437)

Window Styling Guide I

Caption bar (always appear with caption text)

- enabled:
  - if `WS_BORDER` and `WS_DLGFRAME`
  - otherwise if no `WS_POPUP` and no `WS_CHILD`

note: for child windows (i.e. with `WS_CHILD`) top corners of caption bar are rounded unless `WS_EX_TOOLWINDOW` is also specified

Drop-shadow (top-level windows only, i.e. no `WS_CHILD`)

- enabled with `CS_DROPSHADOW`
  note: appears to vanish sometimes when activating or deactivating certain windows

Sunken inner edge

- enabled by `WS_EX_CLIENTEDGE`; exceptions:
  - top-level window with thick border

Window Styling Guide II

Sunken outer edge

- enabled by `WS_EX_STATICEDGE`; exceptions:
  - top-level window with thick frame
  - window with caption bar
  - window with `WS_EX_DLGMODALFRAME`

Raised outer edge

- enabled by `WS_EX_DLGMODALFRAME`; exceptions:
  - top-level window with thick frame
  - window with caption bar
- enabled by `WS_DLGFRAME`; exceptions:
  - top-level window with thick frame
  - window with caption bar
  - window with sunken outer edge

Window Styling Guide III

Raised outer edge (cont'd)

- enabled by `WS_SIZEBOX` (child windows only, i.e. requires `WS_CHILD`); exceptions:
  - window with caption bar
  - window with sunken outer edge
- although mentioned in documentation `WS_EX_WINDOWEDGE` doesn't seem have any discernable effect on this or any other frame visual element or behaviour.

Thick border

- enabled by `WS_SIZEBOX` (border has caption bar color for top-level)
- enabled with caption bar (border has caption bar color)

  note: for top-level windows only top border is visible. Other edges, even if invisible, can be clicked to activate the window. Also the drop-shadow appears outside of the invisible thick frame. Small outer shadow is placed above top border and in place of other edges. Thin border along other edges appears only when window is active.

  note: for top-level windows without `WS_EX_TOOLWINDOW`, caption bar buttons (if present) extend into the top border.

Window Styling Guide IV

Thin border

- enabled by `WS_BORDER`; exceptions:
  - top-level window with thick frame
  - window with caption bar
  - window with sunken inner edge
  - window with raised outer edge
- enabled for windows with sunken outer edge by `WS_DLGFRAME`; exceptions:
  - window with sunken inner edge

Appearance on task bar, in Alt + ⇤⇥, ⊞ + ⇤⇥ windows (top-level windows only, i.e. no `WS_CHILD`)

- always enabled with `WS_EX_APPWINDOW`
- otherwise enabled by default; exceptions:
  - owned window
  - window with `WS_EX_NOACTIVATE`
  - window with `WS_EX_TOOLWINDOW`

Window Styling Guide V

System menu icon

- enabled for windows with caption bar and `WS_SYSMENU`; exceptions:
  - window with `WS_EX_TOOLWINDOW`
  - window with `WS_EX_DLGMODALFRAME`

Close button (windows with caption bar only)

- enabled with `WS_SYSMENU`
  note: button disabled with `CS_NOCLOSE`
  note: top-level windows (i.e. without `WS_CHILD`) that don't have any additional caption bar buttons, the close button is narrower
  note: `WS_EX_TOOLWINDOW` makes close button even smaller (that also affects child windows

Minimize button (only for windows with caption bar and `WS_SYSMENU`)

- enabled with `WS_MINIMIZEBOX` or `WS_MAXIMIZEBOX`; exceptions:
  - window with `WS_EX_TOOLWINDOW`

  note: button disabled without `WS_MINIMIZEBOX`

Window Styling Guide VI

Maximize button (only for windows with caption bar and `WS_SYSMENU`)

- enabled with `WS_MINIMIZEBOX` or `WS_MAXIMIZEBOX`; exceptions:
  - window with `WS_EX_TOOLWINDOW`

  note: button disabled without `WS_MAXIMIZEBOX`

Contextual help button (only for windows with caption bar and `WS_SYSMENU`)

- enabled with `WS_EX_CONTEXTHELP`; exceptions:
  - window with `WS_EX_TOOLWINDOW`, `WS_MINIMIZEBOX` or `WS_MAXIMIZEBOX`

Moving a window by dragging the caption bar:

- enabled for windows with caption bar

  note: for child windows (i.e. with `WS_CHILD`), cursor constrained to parent's client area while dragging

Window Styling Guide VII

Resizing a window by dragging the edges:

- enabled for windows with `WS_SIZEBOX`; constraints:
  - minimum height is equal to sum heights of bottom and top borders and the height of caption bar (if present)
  - for windows with `WS_EX_TOOLWINDOW`, caption bar and no close button, minimum width = width of left and right border plus a small margin
  - for windows with `WS_EX_TOOLWINDOW` and a close button, minimum width from previous case increased by button width
  - for other windows with caption bar, or just `WS_BORDER` or `WS_DLGFRAME`, minimum width = width of left and right border plus the widths of close, minimize, maximize buttons and system icon
    note: caption bar or its elements don't even have to be present.

    note: Measurement appears to be based on child window's caption bar elements — buttons of and pop-up windows, begin far larger, can hang off the left edge.
  - for all other windows minimum width is the sum of widths of left and right border

note: for child windows, cursor constrained to parent's client area while dragging

Combined Styles

Some styles are just combination of others often used in conjunction

```
//Window Styles
           WS_CAPTION == WS_BORDER | WS_DLGFRAME
   WS_OVERLAPPEDWINDOW == WS_CAPTION | WS_SYSMENU | WS_SIZEBOX |
                          WS_MINIMIZEBOX | WS_MAXIMIZEBOX
        WS_POPUPWINDOW == WS_POPUP | WS_BORDER | WS_SYSMENU

//Extended Window Styles
   WS_EX_PALETTEWINDOW == WS_EX_WINDOWEDGE | WS_EX_TOOLWINDOW | WS_EX_TOPMOST
WS_EX_OVERLAPPEDWINDOW == WS_EX_WINDOWEDGE | WS_EX_CLIENTEDGE
```