Win32/64-Programmierung


```
WinMain (/*…*/)
{ MSG      msg;
  WNDCLASS wndclass;
/*wndclass.lpfnWndProc= …
  wndclass.lpszClassName=…*/
  RegisterClass(&wndclass);
  hwnd=CreateWindow(/*…*/)
  ShowWindow (/*…*/);
  UpdateWindow (hwnd);
  while(GetMessage(/*…*/))
  { TranslateMessage(&msg);
    DispatchMessage (&msg);
  } return msg.wParam;
}
```

Windows-Hauptprogramm mit:

- Festlegg d. Fensterklasse, insb.
  - der „Fenster-Prozedur“ und
  - der Klassenbezeichnung
- Anmeldung der Fensterklasse
- Initialisierung der Anwendung
- Eintritt in die Nachrichten-Warteschleife mit
  - Interpretation und
  - Übergabe jeder Nachricht

```
CALLBACK WndProc (/*…*/)
{ switch (/*msg.message*/)
  { case WM_/*…*/:
} }
```

„Fenster-Prozedur“ mit

- Behandlung aller Ereignisse (bzw.Überlassung an Windows)

Win32/64-Programmierung - Beispiel:


```
/*   HelloWin.c   */
#include <windows.h>  /*Es folgt: Vorausdeklaration v.WndProc*/
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   PSTR szCmdLine, int iCmdShow)
{ static TCHAR    szAppName[]=TEXT("HelloWin");
         HWND     hwnd;
         MSG      msg;
         WNDCLASS wndclass; /* Festlegung der Fensterklasse: */
  wndclass.style         = CS_HREDRAW|CS_VREDRAW;//neu zeichnen
  wndclass.lpfnWndProc   = WndProc;//Fenster-Prozedur(Callback)
  wndclass.cbClsExtra    = 0;     //Zusatzspeicher-Reservierung
  wndclass.cbWndExtra    = 0;     //Zusatzspeicher-Reservierung
  wndclass.hInstance     = hInstance;//(aus WinMain-Par.-Liste)
  wndclass.hIcon         = LoadIcon(NULL,IDI_APPLICATION);
  wndclass.hCursor       = LoadCursor(NULL,IDC_ARROW);
  wndclass.hbrBackground =(HBRUSH)GetStockObject(WHITE_BRUSH);
  wndclass.lpszMenuName  = NULL;
  wndclass.lpszClassName = szAppName;   //Name d. Fensterklasse
```

Win32/64-Programmierung - Beispiel:


```
  /* Registrierung der Fensterklasse: */
  RegisterClass(&wndclass); //Initialisierte wndclass=Standard

  /* Initialisierung der Anwendung: */
  hwnd =  CreateWindow(szAppName,TEXT("SysProg goes Windows"),
                       // Name Fensterklasse,      Fenster-Titel
                WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,CW_USEDEFAULT,
                       // Stil,        x-Position,   y-Position,
                         300, 100, NULL, NULL, hInstance, NULL);
             //Breite,Hoehe,hwndEltern,hMenue,Programm-ID,sonst
  ShowWindow (hwnd, iCmdShow);    //Titelleiste und Rahmen
  UpdateWindow (hwnd);            //Fenster fuellen

  /* Nachrichtenschleife: */
  while(GetMessage(&msg,NULL,0,0))//Solange Nachricht!=WM_QUIT
  { TranslateMessage(&msg);       //..interpretieren..
    DispatchMessage (&msg);       //..und Callback uebergeben
  }
  return msg.wParam;            //Parameter der Beendigung
}
```

Win32/64-Programmierung - Beispiel:


```
LRESULT CALLBACK WndProc (HWND hwnd,UINT message,WPARAM wParam,
                                                  LPARAM lParam)
{ HDC         hdc;   // Geraetekennung
  PAINTSTRUCT ps;    // Art d.Zeichenvorgangs
  RECT        rect;  // Zeichenflaeche
  switch (message)
  { case WM_PAINT:                        // Fenster neufuellen:
         hdc=BeginPaint (hwnd, &ps);   // Fenster loeschen
         GetClientRect  (hwnd, &rect); // Flaeche ermitteln
         TextOut (hdc, 0, 0, TEXT("Hello, WinWorld!"), 16);
         EndPaint (hwnd, &ps);         // Geraet-Freigabe
         return 0;
    case WM_DESTROY:    //Ctrl-F4,Alt-F4,[x]-Klick, Systemmenue
         PostQuitMessage(0); // setzt WM_QUIT in Warteschlange
         return 0;
  }
  return DefWindowProc(hwnd,message,wParam,lParam); //sonstiges
}
```

darf nicht fehlen!

HelloWin.exe

```
#include <windows.h>
```

Wesentlicher Inhalt von **`windows.h`**:
5 Include-Dateien mit sämtlichen Konstanten, Funktionen, Daten- und Strukturtypen:

- **`windef.h`** grundlegende Datentypen
- **`winnt.h`** Datentypen für Unicode-Unterstützung
- **`winbase.h`** Funktionen des Systemkerns (engl. *kernel*)
- **`winuser.h`** Funktionen der Benutzerschnittstelle
- **`wingdi.h`** Funktionen der grafischen Geräteschnittstelle

Graphics Device Interface

Windows-UI u. -API


```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE
hPrevInstance, PSTR szCmdLine, int iCmdShow)
```

- **WinMain** (Deklaration: **winbase.h**): Reservierung und Initialisierung von Ressourcen (durch Laufzeitbibliothek vorgegeben – vgl. **main()**)
- **hInstance, hPrevInstance**: **h**andle (d.h.: interner Name, Kennziffer – vgl. **fopen**), vom System zugeteilt als Querverweis auf Daten, die das System in Eigenregie verwaltet (32 Bit, vorzeichenlos).

(Typ als Präfix: „ungarische Notation“)

- **hInstance**: Identifikation der geladenen Programmkopie (“Wer druckt gerade?” “Wessen Fenster überdecke ich gerade?”)
- **hPrevInstance**: Identifikation der zuerst gestarteten Programmkopie – nur für 16-Bit-Programme, wo Kopien Ressourcen teilen: Menü-Texte, Icons etc. (32-Bit-Progr. laufen separiert: **hPrevInstance=0**)
- **PSTR** (**P**ointer to **STR**ing).
- **szCmdLine** ("**s**tring terminated with a **z**ero"): Kommandozeilen-Parameter (vgl. **main (int argc, char *argv[])**)
- **iCmdShow**: **i**nt-Kennzahl für Fensterdarstellung (maximiert, ...)

Präfixe der ungarischen Notation:

b	BOOL (`int`)
br	brush: Füllmuster
by	BYTE (`unsigned char`)
c	`char`, `WCHAR` oder `TCHAR`
cb	„count of bytes“: Größe in Bytes
cx,cy	„count x or y“ (`int`): Koordinaten-Länge
dw	DWORD (`unsigned long`)
f	"flag“: BOOL (`int`)

fn	function
h	handle (`unsigned long`)
i	`int`
l	`long`
n	`short`
p	pointer
s	string
sz	string terminated by zero
w	WORD (`unsigned short`)
x,y	x-, y-Koordinate (`int`)

Präfixe numerischer Konstanten aus `windows.h` :

		z.B.:
CS	Class style option	`CS_VREDRAW`
CW	Create window option	`CW_USEDEFAULT`
DT	Draw text option	`DT_CENTER`
IDI	ID number for an icon	`IDI_APPLICATION`
IDC	ID number for a cursor	`IDC_ARROW`
MB	Message box options	`MB_ICONERROR`
SND	Sound option	`SND_FILENAME`
WM	Window message	`WM_CREATE`
WS	Window style	`WS_OVERLAPPEDWINDOW`

Allgemein (auch) bei Windows:

- **`#define`**-Bezeichner statt Zahlen
- Eigene Datentyp-Bezeichner, meist zur Kompatibilität mit Folge-Versionen - z.B.:

  `UINT` für **`unsigned int`**

  `LRESULT` für Funktionsergebnisse vom Typ **`long`** (32 Bit)

  `LPARAM` für **`long`**-Parameter (Win3x u. Win32: 32 Bit)

  `WPARAM` für **`WORD`**-Parameter (Win3x: 16 Bit, Win32: 32 Bit)

  `WCHAR` für „wide“ **`CHAR`** (Unicode, 16 Bit - ab WinNT)

  `TCHAR` für „text“ **`CHAR`** (16 Bit für Unicode, 8 Bit sonst)
- Umstellung auf Unicode automatisch –interessant dabei:

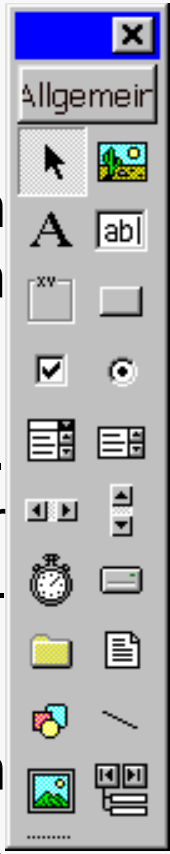
  verwendete Typen (generisch - z.B.: **`WNDCLASS`**) sind meist als ANSI- *und* wide-Versionen definiert (Definitionen entsprechend unter **`WNDCLASSA`**, **`WNDCLASSW`** zu suchen).

`HelloWin.c` erzeugt ein Fenster.

Fenster-Definition:

Rechteckiger Bildschirm-Bereich, d. Benutzer/in-Eingaben entgegennimmt und/oder Ausgaben in Form von Texten und Grafiken anzeigt.

- Schaltflächen (Buttons), Bildlaufleisten, Eingabefelder etc. sind ‘Fenster’ – oft einem Anwendungs- o. Dialogfenster untergeordnet: (Child- bzw.) Kind-Fenster eines (Parent- bzw.) Eltern-Fensters.
- Fenster dienen der Kommunikation zwischen Plattform (hier: Windows) und Anwender/in bzw. Anwendung (MMI, seltener: Sw-Sw-Schnittstelle).



Man-Machine-Interface

Windows-UI u. -API

➡ Die Kommunikation der Applikationsseite (Mensch bzw. App.-Sw) mit der Plattform betrifft das Erzeugen, Manipulieren oder Schließen von Fenstern, Dateien etc.

Dies erfolgt über Objekte (Buttons, Textboxen, ...) bzw. über entsprechende Programmaufrufe (z.B. **`MessageBox`**), welche die Plattform zur Verfügung stellt.

⬅ Die Kommunikation der Plattform mit der App/seite betrifft die Umordnung des Fensterinhalts nach Änderung der Fenstergröße, Maßnahmen nach einem Button-Klick u.a. App.-Aktionen, die der Plattform i.d.R. unbekannt sind.

Dazu sendet die Plattform dem App.-Fenster Nachrichten

– d.h., sie ruft die entsprechende App.-Funktion auf – Windows: die sog. (Window- o.) Fenster-Prozedur.

Callback!

Jedes Windows-Programm / jeder Thread

oft auch: „Ereignis“-Warteschlange

- bekommt beim Start eine eigene **Nachrichten-Warteschlange** (*message queue*) zugewiesen

  – das ist ein systemeigener Bereich, der Nachrichten an die Fenster dieses Programms zwischenspeichert.

oft auch: „Ereignis“-Warteschleife

- enthält eine **Nachrichten-Warteschleife** (*message loop*)

  – sie fragt die Warteschlange ab und verteilt anstehende Nachrichten auf die programmeigenen Fenster.

WinMain(): Fensterklasse

Fensterklassen-Definition in **winuser.h**:

```
typedef struct
{ UINT        style;                    /*Klassenstil*/
  WNDPROC     lpfnWndProc;            /*Fenster-Prozedur*/
  int         cbClsExtra;   /*zusaetzl.Speicher f.Klasse*/
  int         cbWndExtra;     /*zus.Speicher für Fenster*/
  HINSTANCE   hInstance;    /*Handle auf Programm selbst*/
  HICON       hIcon;                  /*Icon-Festlegung*/
  HCURSOR     hCursor;          /*Maus-Cursor-Festlegung*/
  HBRUSH      hbrBackground;        /*Hintergrund-Farbe*/
  LPCSTR      lpszMenuName;                 /*Menue-ID*/
  LPCSTR      lpszClassName;     /*Name d. Fensterklasse*/
} WNDCLASS;
```

WinMain(): Fensterklasse


```
WNDCLASS wndclass;  /*Definition vor Registrierung*/
```

- `UINT style;`
  `wndclass.style = CS_HREDRAW | CS_VREDRAW;`

  ⇨ Setzt den Fenster-Klassenstil mit bitweisem OR (`|`);

  ⇨ Kombination von max. 32 Einzel-(Bit-)Einstellungen;

  ⇨ (Masken:) „Bitflag“-Definitionen in **`winuser.h`**;

  ⇨ hier: `0x0001 | 0x0002 = 0x0003`
  – d.h. Fenster neu zeichnen bei Änderung **H**orizontaler oder **V**ert. Maße (vgl. Text-Umbruch: Textverarbeitung ⇔ Email).

- `WNDPROC lpfnWndProc;`
  `wndclass.lpfnWndProc = WndProc;`

  ⇨ 32-Bit-Zeiger (**`lp`**) auf Callback-Funktion
  (Fenster-Prozedur „win prock“)

`WinMain()`: Fensterklasse

- `int cbClsExtra, cbWndExtra;`
  `wndclass.cbClsExtra = 0;`
  `wndclass.cbWndExtra = 0;`

  ⇨ Nutzung: Reservierung von zusätzlichem Speicher in der Fenster-Klasse bzw. im Fenster.

  ⇨ hier: Nicht-Nutzung: **`= 0;`**

32 Bit als „Platz für Notizen“ (besondere Fenster-Arten)

- `HINSTANCE hInstance;`
  `wndclass.hInstance = hInstance;`

  ⇨ Instanz-Handle auf das Programm selbst (aus Parameterliste von `WinMain` übernommen)

WinMain(): Fensterklasse

- `HICON hIcon;`
  `wndclass.hIcon =LoadIcon(NULL,IDI_APPLICATION);`

  ⇨ Legt Icon für Programm-Titelleiste und -Taskleiste fest

  ⇨ hier: **`LoadIcon`** erwartet **`NULL`** für vordefinierte Icons, **`IDI_APPLICATION`** für Fenster-Logo

- `HCURSOR hCursor;`
  `wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);`

  ⇨ Legt Icon für Mauscursor fest

  ⇨ hier: **`LoadCursor`** erwartet **`NULL`** für vordefinierten Cursor, **`IDC_ARROW`** für Pfeil-Zeiger

WinMain(): Fensterklasse

- `HBRUSH hbrBackground;`
  `wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);`
  - ⇨ Legt Fenster-Hintergrundfarbe o. -muster fest
  - ⇨ hier: Weiß (Brush=Dicker Pinsel; Stock=Lager)

- `LPCSTR lpszMenuName;`
  `wndclass.lpszMenuName = NULL;` `/* =(LPCSTR)IDR_MENU; */`
  - ⇨ Legt das Fensterklassenmenü fest (32-Bit-Zeiger auf Zeichenkette).
  - ⇨ Für Fensterklassen ohne Menü: **`=NULL`**
  - ⇨ Mit Menü: entsprechende ID auf **`(LPCSTR)`** casten

WinMain(): Fensterklasse

- `LPCSTR lpszClassName;`
  `wndclass.lpszClassName = TEXT ("HelloWin");`
  - ⇨ Name der Fensterklasse (beliebig);
  - ⇨ Kennzeichnet die Fensterklasse innerhalb der Applikation;
  - ⇨ Falls keine Verwechslungsgefahr (z.B.: Ein-Fenster-App.): meist gleich dem Programm-Namen gewählt.

sichert Wechsel ANSI / Unicode

- Registrieren der Fenster-Klasse mit **`RegisterClass`**
  Parameter: Zeiger auf die initialisierte **`WNDCLASS`**-Struktur
  `RegisterClass (&wndclass);`

  [Prototyp:

```
WINUSERAPI ATOM WINAPI RegisterClassA (CONST WNDCLASSA
                                       *lpWndClass);]
```

unsigned short

DLLs einbeziehen

Compilieren ohne Optimierung

  - ⇨ Sorgt auch für korrekte Übermittlung von Zeichen-Nachrichten (ANSI ⇔ Unicode)

WinMain(): Initialisierung

Funktionsprototyp zum Einrichten (Anlegen) von Fenstern:

```
HWND CreateWindow(      /*Rueckgabe: Fenster-Handle*/
  LPCTSTR lpClassName,/*registr.Fensterklasse-Name*/
  LPCTSTR lpWindowName,             /*Fenstertitel*/
  DWORD dwStyle,                     /*Fensterstil*/
  int x,              /*Fenster-X-Position (Pixel)*/
  int y,              /*Fenster-Y-Position (Pixel)*/
  int nWidth,              /*Fensterbreite (Pixel)*/
  int nHeight,             /*Fensterhoehe  (Pixel)*/
  HWND hWndParent,       /*uebergeordnetes Fenster*/
  HMENU hMenu,                       /*Menue-Handle*/
  HINSTANCE hInstance, /*Prog.Kopiezaehler(ProgID)*/
  LPVOID lpParam          /*zusaetzliche Parameter*/
 );
```

WinMain(): Initialisierung

Anlegen eines Fensters mit **CreateWindow**:

```
HWND   hwnd;                        /*Handle to a window*/
hwnd = CreateWindow(  /*Rueckgabe: Fenster-Handle*/
   TEXT("HelloWin"), /*registr.Fensterklasse-Name*/
   TEXT("SysProg goes Windows"),   /*Fenstertitel*/
   WS_OVERLAPPEDWINDOW,             /*Fensterstil*/
   CW_USEDEFAULT,    /*Fenster-X-Position (Pixel)*/
   CW_USEDEFAULT,    /*Fenster-Y-Position (Pixel)*/
   CW_USEDEFAULT,         /*Fensterbreite (Pixel)*/
   CW_USEDEFAULT,         /*Fensterhoehe  (Pixel)*/
   NULL,                /*uebergeordnetes Fenster*/
   NULL,                           /*Menue-Handle*/
   hInstance,          /*Prog.Kopiezaehler(ProgID)*/
   NULL);                 /*zusaetzliche Parameter*/
```

WinMain(): Initialisierung

- **(TEXT("HelloWin"),** /*Fensterklasse-Name*/
  - ⇨ Verbindung zw. Fenster u. Fenster-Klasse

- **TEXT("SysProg goes Windows"),** /*Fenstertitel*/
  - ⇨ Beschriftung der Titelleiste

- **WS_OVERLAPPEDWINDOW,** /*Fensterstil*/
  - ⇨ Zuordnung von Systemmenü (o.li.), Schaltflächen (o.re.), ...
  - ⇨ „Bitflag“-Definitionen in **winuser.h**:

```
#define WS_OVERLAPPEDWINDOW \
    (WS_OVERLAPPED   | \    /*Ueberlappend*/
     WS_CAPTION      | \/*Titelleiste-Text*/
     WS_SYSMENU      | \     /*Systemmenue*/
     WS_THICKFRAME   | \   /*dicker Rahmen*/
     WS_MINIMIZEBOX | \      /*Minimieren*/
     WS_MAXIMIZEBOX)         /*Maximieren*/
```

WinMain(): Initialisierung

- **CW_USEDEFAULT,CW_USEDEFAULT,** /*X/Y-Position*/
  CW_USEDEFAULT,CW_USEDEFAULT,/*F.breite,-hoehe*/
  ⇨ standardgroße Fenster in Treppenformation - s. **winuser.h**

- **NULL,NULL,** /*kein Eltern-Fenster, kein Menue*/
  ⇨ Desktop gilt nicht als übergeordnet,
  System-Menü zählt nicht als Menü

- **hInstance,** /*Programm-Kopiezaehler(Prog.-ID)*/
  ⇨ aus Parameterliste von WinMain

- **NULL);** /*Erstellungsparameter*/
  ⇨ kein Zugriff auf reservierten Bereich

Ergebnis von **CreateWindow**:
Fenster als Datenstruktur, identifizierbar über **hwnd**

`WinMain()`: Initialisierung

Fenster sichtbar machen mit:

- **`ShowWindow (hwnd, iCmdShow);`**

  `hwnd`: Rückgabewert von **`CreateWindow`**
  `iCmdShow`: aus der Parameterliste von **`WinMain`**, z.B.:
  `SW_SHOWNORMAL, SW_SHOWMAXIMIZED,`
  `SW_SHOWMINNOACTIVE, SW_HIDE`

  ⇨ Zeichnet Titelleiste und Fensterrahmen

Fensterinhalt anzeigen:

- **`UpdateWindow (hwnd);`**

  ⇨ Schickt Nachricht des Typs **`WM_PAINT`** an die Fenster-Prozedur (hier: **`WndProc`**)

WinMain(): Nachr.-Warteschleife

- Nachrichten-Warteschleife:

```
while (GetMessage  (&msg, NULL, 0, 0))//holen..
{ TranslateMessage (&msg); //..interpretieren..
  DispatchMessage  (&msg); //..uebergeben
} return msg.wParam; //(Parameter d.Beendigung)
```

- **msg-**Def. (**winuser.h**):

```
typedef struct
{ HWND      hwnd;
  UINT      message;
  WPARAM    wParam;
  LPARAM    lParam;
  DWORD     time;
  POINT     pt;
} MSG;
```

je 4Byte

- **POINT-**Def. (**winuser.h**):

```
typedef struct
{ LONG  x;
  LONG  y;
} POINT;
```

WinMain(): Nachr.-Warteschleife

- Der Aufruf
  GetMessage(&msg, NULL, 0, 0);
  liest die jeweils nächste Nachricht in der Warteschlange.
  ("**NULL,0,0**": Alle Nachrichten für alle Fenster der Applikation)
  Für Nachricht **WM_QUIT** ist der Rückgabewert 0.
  Der Lesevorgang setzt die msg-Variablen:
  msg.hwnd: Handle zum Fenster des Nachricht-Empfängers
  (zuerst zugewiesen durch **CreateWindow**)
  msg.message: **UINT**-Konstante als Nachricht-Codierung
  (Button-Klick etc. - Defin. in **winuser.h**)
  msg.wParam: **WORD**-Parameter (32-Bit; früher: 16-Bit)
  msg.lParam: **LONG**-Parameter (32-Bit; ereignisabhängig)
  } Bedeutung ereignisabhängig
  msg.time: Uhrzeit zum Zeitpunkt des Ereignisses
  msg.pt: Cursor-Koordinaten z.Z. des Ereignisses

[Prototyp:

```
WINUSERAPI BOOL WINAPI GetMessageA (LPMSG lpMsg,
   HWND hWnd, UINT wMsgFilterMin, UINT wMsgFilterMax);]
```

WinMain(): Nachr.-Warteschleife

- Die von **GetMessage** neubesetzte **msg**-Struktur wird mit
  TranslateMessage (&msg);
  an Windows übergeben (Interpretation von Tastaturbefehlen).
  `[WINUSERAPI BOOL WINAPI TranslateMessage (CONST MSG *lpMsg);]`

- Die Anweisung
  DispatchMessage (&msg);
  ermittelt die für ein Ereignis zuständige (applikationseigene) Fenster-Prozedur und leitet die Nachricht an sie weiter.
  `[WINUSERAPI LONG WINAPI DispatchMessageA (CONST MSG *lpMsg);]`

- Der Rücksprung
  return msg.wParam;
  erfolgt, falls **GetMessage** in der Warteschlange **WM_QUIT** liest;
  das kann auch von der Fenster-Prozedur ausgelöst werden (Aufruf **PostQuitMessage(0)** setzt nächste Nachricht der Warteschlange auf **WM_QUIT**).

WndProc

- Die Fenster-Prozedur:

long Compiler-Anweisg. bel. Name

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT message,
                          WPARAM wParam, LPARAM lParam);
```

hat vier Parameter, die mit den ersten vier Variablen von msg übereinstimmen:

hwnd: Handle zum Empfänger-Fenster der Nachricht

message: Nachricht-Codierung (**WM_**... , Def. in **winuser.h**)

wParam: 32-Bit-Parameter (Bedeutung ereignisabhängig)

lParam: wie **wParam** (**wParam** früher: 16-Bit-**WORD**)

- Erster **WndProc**-Aufruf aus **CreateWindow** mit **WM_CREATE** - noch vor Aufbau d. Warteschlange (nutzbar für Initialisierungen)
- Letzter Aufruf (z.B. aus **DestroyWindow()**) mit **WM_DESTROY**; Reaktion mit Aufruf von **PostQuitMessage(0);** (⇨ **WM_QUIT**)

WndProc

- Die Reaktion der Fenster-Prozedur auf die erhaltene Nachricht erfolgt meist in einer **switch**-Anweisung mit:

```
switch (message)
{
  case WM_CREATE:
       /* Erste Nachricht -noch ohne Warteschlange-
          Platz fuer Initialisierungen, Vorspann..*/
       return 0;/* =OK, sonst -1*/
  case WM_PAINT:
       /* WM_PAINT - Behandlung */
       return 0;
  case WM_DESTROY:
       /* Letzte Nachricht- WM_DESTROY-Behandlung */
       return 0;
}
return DefWindowProc(hwnd, message, wParam, lParam);
```

WM_*: Window Message

default window procedure

WndProc

- Aufruf der Fenster-Prozedur mit **WM_PAINT** bei Fenster-
  - -Erzeugung
  - -Größen- o. Inhaltsänderung
  - -Ikonifizierung
  - -Aufdeckung

  } Fenster-Inhalt „ungültig“ (*invalid*)

- Beispiel für **WM_PAINT**-Behandlung:

unsigned long

handle to device context

Weniger gebräuchlich: **DrawText()**

Anzahl Zeichen

```
HDC hdc;  PAINTSTRUCT ps;  RECT rect; /*...*/
case WM_PAINT:
hdc = BeginPaint(hwnd,&ps);  /*Fenster loeschen*/
GetClientRect(hwnd,&rect);   /*Zeichenflaeche*/
TextOut (hdc,0,0,TEXT("Hello, WinWorld!"),16);
EndPaint(hwnd,&ps);          /*hdc-Freigabe*/
return 0;
```

WndProc

Datentyp-Definitionen:

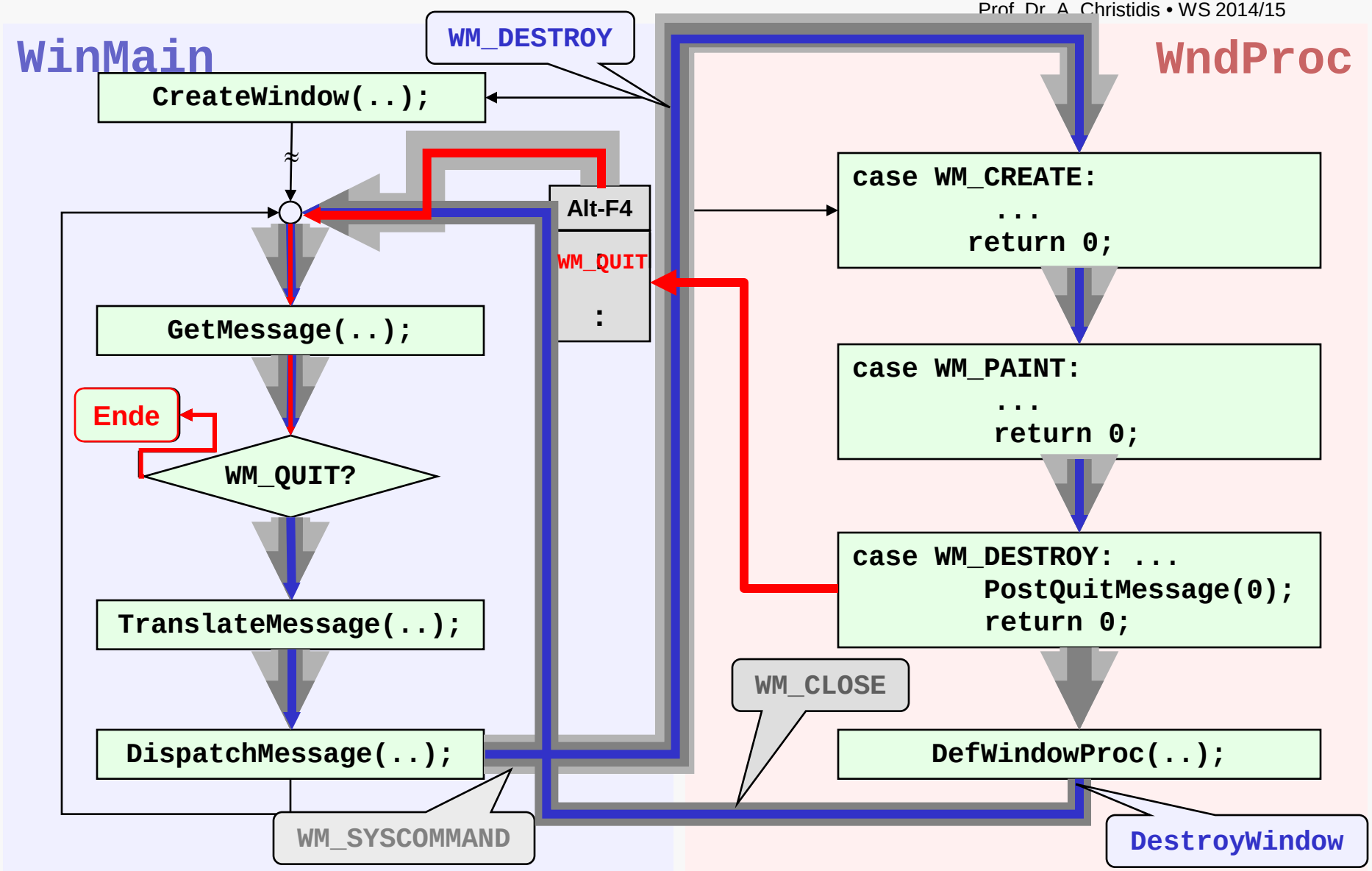
```
typedef struct tagRECT
{ LONG    left;
  LONG    top;
  LONG    right;
  LONG    bottom;
} RECT;


typedef struct tagPAINTSTRUCT
{ HDC     hdc;
  BOOL    fErase;
  RECT    rcPaint;
  BOOL    fRestore;
  BOOL    fIncUpdate;
  BYTE    rgbReserved[32];
} PAINTSTRUCT ;
```

FALSE, wenn Fenster-Hintergrund bereits gelöscht (= aktualisiert)

3 Variablen für Windows-interne Verwendung

Win32-Programmierung – Beispiel:



Win32-Programmierung

Vorläufige „Erkenntnisbilanz“:

- Windows-Programme „laufen nicht ab“: Sie reagieren auf Nachrichten, die ihrerseits (u.a.) von Eingaben generiert werden.
- Hauptgeschehen in der Fenster-Prozedur
- „Indirekte“ Nachrichten an Fenster-Prozed. werden in d. N.-Warteschlange „eingereiht“ (*queued* / **`DispatchMessage`**), „direkte“ von Windows an Fenster-Prozedur „gesendet“ (*nonqueued* / **`CreateWindow`**)
- Forderung: Windows - Programme „reentrant“ („wiedereintritts - invariant“): Mehrfach-Aufrufe der Fenster-Prozedur - z.B. über **`DefWindowProc`**. (⇨ `static` **`!`**)

Erkenntnisse mit SP.c:

- Ereignis-Orientierung: App.c abhängig von Interaktion mit SP.c
- „Fachlich relevanter Programmteil“ i.App.c (**`redisplay`** / **`calc`**)
- Callback - Aufrufe (**`redisplay`**) nach Tastendruck (‘s‘/‘d‘) u. für Timer-Ausgabe
- **`SPprintf`**
  ⇨ aus **`redisplay`**
  ⇨ aus **`SPprintf`**