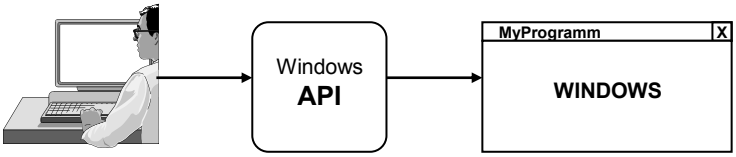
Arbeiten mit der Windows API
(Application Programming Interface)

1. Was ist eine API (Application Programming Interface)?

Um ein Windows Programm zu implementieren muss der Softwareentwickler nicht das ganze Betriebssystem programmieren.

☞ **Erläutern Sie, auf welche Weise ein Programmierer das Betriebssystem für seine Anwendungen nutzen kann!**



Zum Entwickeln von Windowsanwendungen steht dem Softwaretechniker die Windows API (Application Programming Interface) zur Verfügung. Sie dient als Schnittstelle zwischen Anwendung und Betriebssystem.
In der API sind Variablen, Typen, Structures, Konstanten, Funktionen und Makros zur Verwendung unter Windows definiert. Der Programmierer kann über diese vordefinierte Schnittstelle direkt mit dem System zusammenarbeiten (z.B. bei der Erzeugung eines Fensters). Da die API mit der Entwicklung des Betriebssystems gewachsen ist, würde es mehr als ein halbes Jahr dauern, um alle Möglichkeiten der API bis ins Detail kennenzulernen ☹!

Zusätzlich zur API gibt es eine ganze Reihe weiterer Interfaces für spezielle Anwendungen:

- **GDI (Graphics Device Interface)**: wird für einfache, geräteunabhängige Grafikoperationen benutzt,
- **MAPI (Messaging Applications Interface)**: gewährt den Anwendungen Zugriff auf bestimmte Nachrichtensysteme (z.B. Microsoft Mail),
- **MCI (Multimedia Control Interface)**: die Multimedia - Schnittstelle,
- **COM - API**: Sammlung von Systemaufrufen für die OLE Funktionalität,
- **TAPI**: Telefonie - API für den Zugriff auf Modems, Fax...,
- **WinSock**: für Netzwerk Funktionalität,
- **WinInet**: die Windows Internet - API,
- **RAS (Remote Access Service)**: für den Zugriff auf Ressourcen von Außen.

2. Windowsprogramme mit der API

Einige Programmierer, die über die Windows - Programmierung sprechen, sind der Meinung, dass mit einem Betriebssystem etwas nicht stimmen kann, das sehr viele Programmzeilen benötigt, um ein einfaches Programm zur Ausgabe von "Hello World" ausführen zu lassen!

☞ **Erzeugen Sie dazu eine leere Win32 - Anwendung und testen Sie folgendes Programm!**
☞ **Legen Sie als Zeichensatz „Multi-Byte-Zeichensatz verwenden“ bei den Projekteigenschaften fest (Projekt/ Eigenschaften -> Konfigurationseigenschaften, Allgemein, Zeichensatz)!**

2.1 Hallo Windows Welt!

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE d1, HINSTANCE d2, LPSTR d3, int d4)
{
      MessageBox(NULL, "Hallo Windows Welt!", "", MB_OK);
}
```

Gratulation! Sie haben gerade Ihr erstes Windows - Programm mit Ereignissteuerung und grafischer Oberfläche implementiert.

Programmbeschreibung:
Jedes API - Programm wird über die Hauptfunktion WinMain(...) gestartet. WINAPI ist ein Makro und sorgt für die richtige Aufrufkonvention beim Kompilieren.
HINSTANCE d1 ist ein von Windows definierter Datentyp für den Instance - Handle des Fensters, mit d2 kann man unter Win16 überprüfen, ob bereits eine Instanz der Anwendung ausgeführt wird. Unter Win32 ist d2 immer NULL (jede Instanz läuft in einem eigenen Prozess ab!).
LPSTR d3 übernimmt eventuelle Kommandozeilenparameter, int d4 speichert, wie das Fenster angezeigt werden soll. Die Funktion MessageBox(...) erzeugt ein Meldungsdialogfeld.

☞ **Testen Sie MessageBox(...) mit einem anderen String anstelle von ""!**
☞ **Verwenden Sie die Parameter MB_OKCANCEL, MB_RETRYCANCEL, MB_ABORTRETRYIGNORE, MB_YESNO und MB_YESNOCANCEL in Kombination (|) mit MB_ICONEXCLAMATION, MB_ICONINFORMATION, MB_ICONQUESTION oder MB_ICONSTOP!**

Das erste Programm gibt noch keinen Aufschluss über die Funktionalitäten eines echten Windows - Programms. Sowohl die Message Loop als auch die Fensterfunktion sind vor dem Programmierer verborgen.
Beides ist in der Funktion MessageBox(…) gekapselt.

Arbeiten mit der Windows API
(Application Programming Interface)

2.2. Hallo Windows Welt! (2)

Hier werden zwei Variablen für Botschaften (msg) und Fensterhandle (hwnd) deklariert.
Die eigentliche Erzeugung des Fensters findet in der Funktion CreateWindow(…) statt.
Als erster Parameter wird die Fensterklasse "BUTTON" übergeben. Wir erzeugen also einen Schalter.
Eine Fensterklasse darf in Windows nicht mit einer Klasse im objektorientierten Sinn verwechselt werden! Vielmehr handelt es sich um eine Structure, in der die Eigenschaften des Fensters und ein Zeiger auf seine Fensterfunktion definiert sind. Windows arbeitet historisch gesehen also strukturiert mit "objektorientierten" Konzepten.
"Hallo ..." ist die Beschriftung. WS_... sind vordefinierte Windowsstile zur Anzeige, anschließend folgen Parameter zu Größe und Position, zum übergeordneten Fenster, zu einem vorhandenen Menü, zur eigenen Instanz und zu weiteren Erzeugungsdaten. Bis auf den Instanz - Handle sind alle Parameter auf NULL gesetzt.

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE d2, LPSTR d3, int d4)
{
  // Message und Fensterhandle erzeugen
  MSG msg;
  HWND hwnd;

  // Fenster der Kategorie BUTTON erzeugen
  hwnd = CreateWindow("BUTTON", "Hallo Windows Welt!",
              WS_VISIBLE | BS_CENTER, 100, 100, 200, 80,
              NULL, NULL, hInstance, NULL);

  // Start der Nachrichtenschleife
  while (GetMessage(&msg, NULL, 0, 0))
  {
     // Wenn linke Maustaste losgelassen...
     if (msg.message == WM_LBUTTONUP)
     {
        // Fenster zerstören
        DestroyWindow(hwnd);
        // Message Loop verlassen
        PostQuitMessage(0);
     }
     DispatchMessage(&msg);
  }
  return msg.wParam;
}
```

Mit der while - Schleife beginnt die Message Loop. Solange Nachrichten eintreffen werden diese mit DispatchMessage(..) an die Standardfensterfunktion von BUTTON weitergeleitet. PostQuitMessage(0) gewährleistet, dass GetMessage( ) auf die WM_QUIT Nachricht wartet, woraufhin die Message Loop verlassen und das Fenster geschlossen wird. Trifft die Nachricht WM_LBUTTONUP (der gedrückte Button wird wieder losgelassen), so wird das Fenster ebenfalls zerstört.

☞ **Testen Sie andere Parameter für die Größe und Position des Fensters!**
☞ **Nutzen Sie auch die MSDN um Klarheit über die neuen Funktionen zu erhalten!**

Auch diese Version von Hallo Windows Welt verbirgt den Großteil der Funktionalität vor dem Anwender. Wo zum Beispiel ist die Fensterfunktion definiert? Gar nicht! Wir nutzen die Standardfensterfunktion der Windowsklasse BUTTON. Etwas komplexer wird das nächste, "echte" Hallo Windows Welt Programm.

2.3 Hallo Windows Welt! (3)

Die Fensterfunktion der Anwendung (WndProc... Windows Procedure)

Hier wird zunächst die Fensterfunktion definiert. Sie erhält und bearbeitet alle Botschaften, die vom Benutzer an das Fenster geschickt werden.
Anschließend wird ein Gerätekontext (HDC handle Device) für die Grafikausgabe erzeugt.
In einer switch - Anweisung werden die Nachrichten an das Fenster ausgewertet.
Soll das Fenster neu gezeichnet werden, so lädt man den Gerätekontext und der Text wird mit TextOut(...) ausgegeben. EndPaint(...) gibt den Gerätekontext für Windows wieder frei.
WM_DESTROY verläuft wie im vorherigen Programm.
Der default - Zweig gibt alle anderen Nachrichten (z.B. Verschieben, Vollbild... an die Standardfensterfunktion DefWindowProc(...) weiter.
Jedes Fenster in Windows arbeitet mit einer mehr oder weniger umfangreichen Fensterfunktion. Wie komplex die switch - Anweisung ausfällt, ist von der Anzahl der Ereignisse abhängig, auf die ein Fenster reagieren soll. Das können sehr viele sein ☺!

```
// Fensterfunktion für die Anwendung
LRESULT CALLBACK WndProc(HWND hwnd,
    UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  // Gerätekontext für Grafikausgabe
  HDC hDC;
  PAINTSTRUCT ps;
  // Nachrichten auswerten
  switch(uMsg)
  {
     case WM_PAINT:
     hDC = BeginPaint(hwnd, &ps);  //Gerätekontext laden
     if (hDC != NULL)     //Wenn kein Fehler aufgetreten
     {
        char str[30] = "Hallo Windows Welt!";
        TextOut(hDC, 45, 15, str, strlen(str));//Ausgabe
        EndPaint(hwnd,&ps);    //Gerätekontext freigeben
     }
     break;
     case WM_DESTROY:         //Fenster wird geschlossen
     PostQuitMessage(0);
     break;
     //Andere Nachrichten an Standardfensterfunktion!
     default:
     return DefWindowProc(hwnd, uMsg, wParam, lParam);
  }
  return 0;
}
```

Arbeiten mit der Windows API
(Application Programming Interface)

Die Eintrittsfunktion WinMain(...)

Wieder werden Variablen für die Windowsbotschaften und den Fensterhandle deklariert. Neu ist die Variable für eine Fensterstruktur (WNDCLASS).

Wenn noch kein Fenster derselben Fensterklasse bei Windows registriert ist, werden alle Werte der Structure durch memset(…) mit 0en ausgefüllt.
Anschließend werden die Elemente wie horizontales und vertikales Neuzeichnen, Cursor, Hintergrundfarbe und Fenstertext der Structure definiert.

Als Fensterfunktion wird die vorher definierte WndProc gesetzt.
Mit RegisterClass(...) registriert man die Fensterklasse bei Windows. Schlägt die Registrierung fehl, so wird das Programm abgebrochen.

Die Fensterklasse ist nun registriert und kann unter Windows mit CreateWindow(...) erzeugt werden.

Damit das Fenster angezeigt und aktualisiert wird, nutzt man die Funktionen ShowWindow(...) und UpdateWindow(...) der API. Das Fenster wird mit aktuellem Aussehen angezeigt.

Zum Schluss ist wieder die Message Loop zu starten. Wie im letzten Beispielprogramm werden die Nachrichten mit DispatchMessage(…) an die Fensterfunktion WndProc(…) zur Verarbeitung übergeben.

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE
                   hPrevInstance, LPSTR d3, int nCmdShow)
{
   // Message, Fensterhandle und Windowsklasse erzeugen
   MSG msg;
   HWND hwnd;
   WNDCLASS wndClass;

   // Wenn noch keine Instanz des Fensters läuft...
   if (hPrevInstance == NULL)
   {
      // Alle Werte der Fensterklasse mit 0 füllen
      memset(&wndClass, 0, sizeof(wndClass));

      // Windows Fensterklasse beschreiben...
      wndClass.style = CS_HREDRAW | CS_VREDRAW;
      wndClass.lpfnWndProc = WndProc;
      wndClass.hInstance = hInstance;
      wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
      wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
      wndClass.lpszClassName = "HELLO";

      // Fensterklasse kann nicht registriert werden,
      // Abbruch
      if (!RegisterClass(&wndClass))
         return FALSE;
   }

   // Fenster erzeugen
   hwnd = CreateWindow("HELLO", "HELLO WINDOWS WORLD",
                       WS_OVERLAPPEDWINDOW,
                       10, 10, 245, 90,
                       NULL, NULL, hInstance, NULL);

   // Fenster anzeigen
   ShowWindow(hwnd, nCmdShow);
   UpdateWindow(hwnd);

   // Message Loop starten
   while (GetMessage(&msg, NULL, 0, 0))
      DispatchMessage(&msg);

   return msg.wParam;
}
```

Gratulation! Nun haben Sie ihr erstes "wirklich echtes" Windows - Hello World Programm implementiert ☺!

2.4 Der Prinzipielle Aufbau eines Windows - Programms

☞ **Beschreiben Sie den prinzipiellen Aufbau eines Windows - Programms!**

Ein gewöhnliches Windows - Programm registriert in WinMain(...) zunächst eine Fensterklasse während der Initialisierung. Die Fensterklasse ist als Structure vordefiniert und muss nur mit Informationen zum Fenster und seiner Fensterfunktion gefüllt werden.
Anschließend erzeugt man das Fenster mit CreateWindow(...). ShowWindow(...) bzw. UpdateWindow(...) sorgen für die Anzeige.
Zuletzt startet man die Message Loop, damit das Fenster auf Botschaften reagieren kann.
Alle Windows - Botschaften landen in der Fensterfunktion. Hier werden Sie in einer switch - Anweisung ausgewertet. Für Ereignisse, die man selbst beantworten will, implementiert man eigenen Quelltext.
Standardbotschaften (z.B. Verschieben, Größe ändern...) schickt man im default - Zweig an die Standard - Fensterfunktion DefWindowProc(...) des Betriebssystems weiter. Um ihre Verarbeitung muss sich der Programmierer dank der Windows API nicht kümmern.

Windows kennt derzeit etwa 500 verschiedene Nachrichten (ohne .NET Erweiterungen). Glücklicherweise muss der Programmierer nicht selbst für die korrekte Bearbeitung all dieser Nachrichten sorgen. Worauf es letztlich ankommt, ist die für ein Programm wirklich interessanten Nachrichten abzufangen und mit dem Aufruf passender Antwortfunktionen zu verbinden.