# Windows Controls, Dialoge

## Ziel, Inhalt

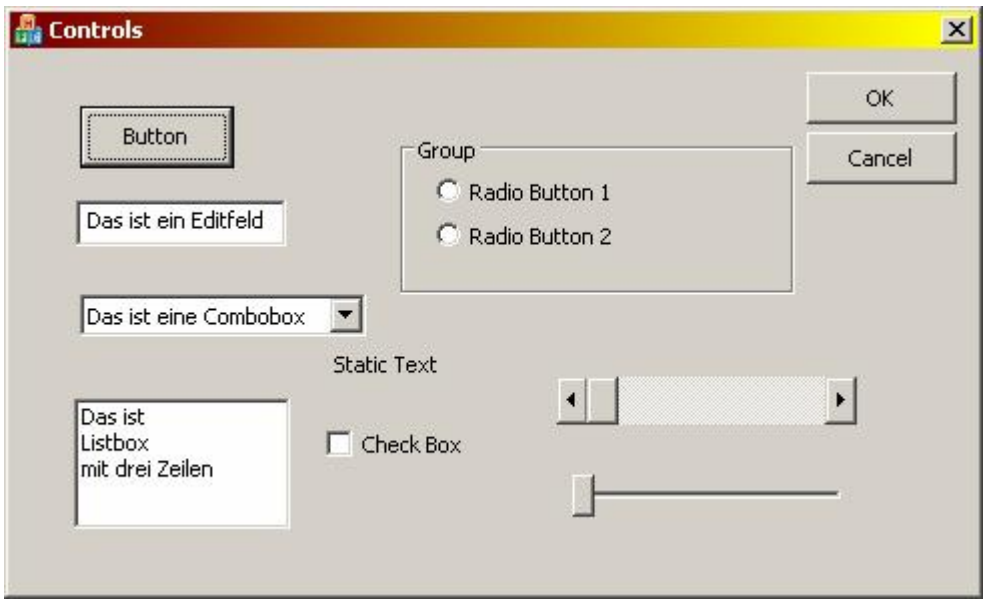
- Das Windows Betriebssystem bietet einige Elemente an, die dazu dienen dem Benutzer Information darzubieten oder vom Benutzer abzufragen. Diese Elemente heissen Controls und wir sind ab heute in der Lage einen Dialog mit solchen Elementen zu erzeugen.

- Solche Controls können durch einen einfachen Eingriff individuell gestaltet werden. In einem Übungsteil lernen wir, wie wir einen Button ein wenig verschönern können.

# Windows Controls

## Überblick

Windows bietet einige Controls an, die wir benutzen um dem Benutzer Information anzuzeigen oder von ihm abzufragen. Die MFC hat diese Standard Controls mit Klassen gekapselt. In einem kleinen Dialogprojekt lernen wir jetzt einige davon kennen.



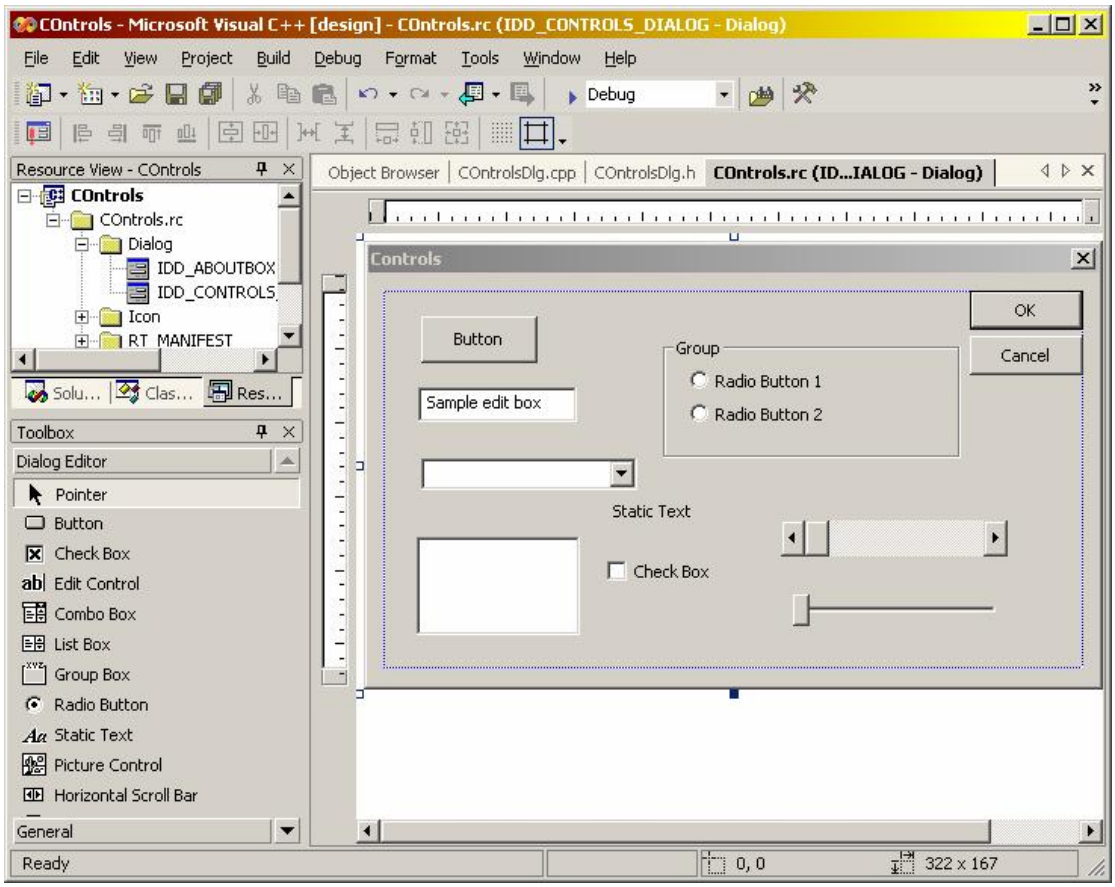

Dialog mit verschiedenen Controls

## Erstellen des Projektes

Ähnlich wie letztes Mal erstellen wir wieder ein neues Projekt mit dem MFC-Applikationsassistenten. Diesmal wählen wir als Projekt eine Dialogbasierende Applikation. Diese Applikationsart erinnert stark an die Form-basierte Entwicklung mit der Borland-Entwicklungsumgebung.

### Bearbeiten des Dialoges

Um den Dialog zu bearbeiten und einige Controls darauf zu platzieren wechsle von der Klassenansicht zu der Ressourcenansicht und klicke doppelt auf den Eintrag IDD_XXX, der deinem Dialog entspricht. Üblicherweise öffnet sich der Ressourceneditor automatisch wenn das Projekt erstellt wird.
Mit dem neuen Visual Studio ergibt sich folgendes Bild (es sollte sich nicht zu stark vom alten Studio unterscheiden)




Der Ressourcen-Editor mit der Toolbox (links)

Ziehe nun einige Controls aus der Toolbox auf den Dialog. Wähle aus den möglichen Controls folgende aus

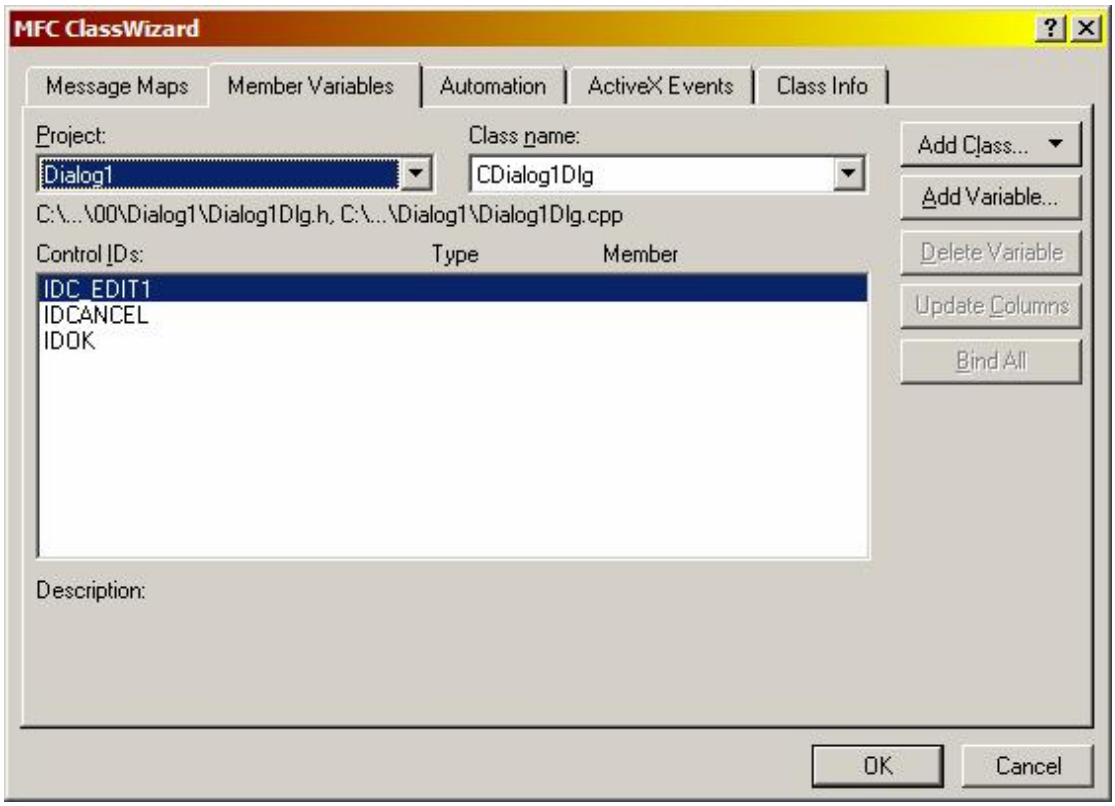
- Button. Dieses Control ist ein Knopf, der gedrückt werden kann
- Check Box. Die Check Box ist in Wahrheit auch ein Button, der besonders dargestellt wird
- Edit Control. Das Edit-Feld wird verwendet um Text in einem Feld darzustellen. Dieser Text kann vom Benutzer geändert werden
- Combo Box. Dieses Control ist sehr vielseitig. Es kann als aufklappende Liste dem Benutzer eine Auswahl anbieten. Der Benutzer kann je nach dem eigene Einträge machen
- List Box. Die List Box ist eine Liste von Text-Einträgen
- Group Box. Diese Element dient zur optischen Gruppierung von anderen Controls.
- Radio Button. Radio Buttons verhalten sich wie Knöpfe auf einem alten Radio. Drückt man den einen, springt der andere wieder heraus.
- Scroll Bar. Dieses Control entspricht dem Control, das verwendet wird um in Fenstern herumzuscrollen.
- Slider Control. Diese Control ist ähnlich zum Scrollbar.

**Die erzeugte Dialogklasse**

Im Sinne eines Frameworks wurde für uns vom Assistenten eine Dialogklasse erzeugt, die von der MFC-Klasse CDialog erbt. Automatisch wurden die Methoden OnPaint und OnInitDialog erzeugt. OnPaint wird aufgerufen wenn der Dialog gezeichnet werden muss. Wir brauchen uns hier aber nicht darum zu kümmern, denn der Dialog und die Elemente darauf können sich selber um das Neuzeichnen kümmern. OnInitiDialog wird aufgerufen wenn der Dialog initialisiert wird. Dort werden wir später ein wenig Code einbauen.

## Verbinden der Controls mit Datenelementen

Im Gegensatz zur Borland-Umgebung werden hier nicht automatisch Datenelemente in der Dialogklasse erzeugt. Der Klassen-Assistent hilft uns aber dabei weiter. Starte nun also den Klassen-Assistenten (Ctrl+W).

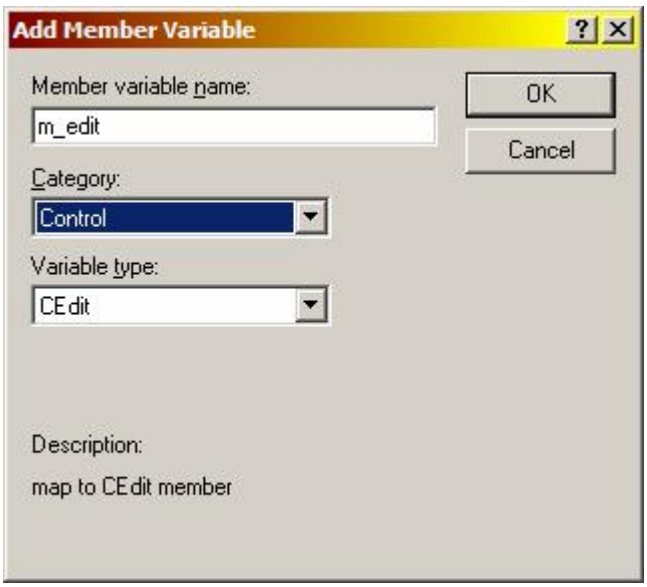



Der Klassenassistent im Visual Studio

Im Tabulator „Member Variables“ erscheinen bei mir für jedes Control, das ich auf den Dialog gezogen habe eine ID. Diese ID kann übrigens mit anderen Eigenschaften eines jeden Elementes mit den Control-Properties geändert werden.


Die Eigenschaften eines Controls, hier mit der ID

Im Assistentendialog können nun die einzelnen Controls gewählt werden und durch einen Klick auf „Variable hinzufügen“ landen wir in einem Dialog wo es möglich ist eine Variable für das Control zu erzeugen.




Dialog zum Einfügen einer Variablen

Dieser Dialog bietet an einen Namen zu wählen und eine Category zu bestimmen. Für gewisse Controls wie dem Edit-Control ist es möglich direkt eine String-Variable zu erzeugen, die dem Text im Control entspricht. Für unseren ersten Versuch verbinden wir aber die Controls mit der entsprechenden Control-Klasse aus der MFC.

### DoDataExchange

Um diese neu erzeugten Datenelemente mit dem entsprechenden Control auf dem Dialog zu verbinden hat der Assistent Code in die Methode DoDataExchange eingefügt, der jeweils etwa so aussieht:

```
DDX_Control(pDX, IDC_EDIT, _edit);
```

### Zum Teufel mit public Datenelementen

Leider ist den Programmierern bei Microsoft damals sauberer C++ Code schnurz-piep-egal gewesen. Alle Elemente werden einfach im public Teil der

Klasse deklariert. Wir verschieben diese einfach in den private Teil! Auch hier ist Kapselung oberstes Gebot.

## MFC Control-Klassen

### Erzeugte Controls

| Control | MFC-Klasse |
|---|---|
| Edit | CEdit |
| Button | CButton |
| Check Box | CButton |
| Combo Box | CCombobox |
| List Box | CListbox |
| Scrollbar | CScrollbar |
| Slider | CSliderCtrl |
| Static | CStatic |

Tabelle mit den Controls und den entsprechenden MFC-Klassen

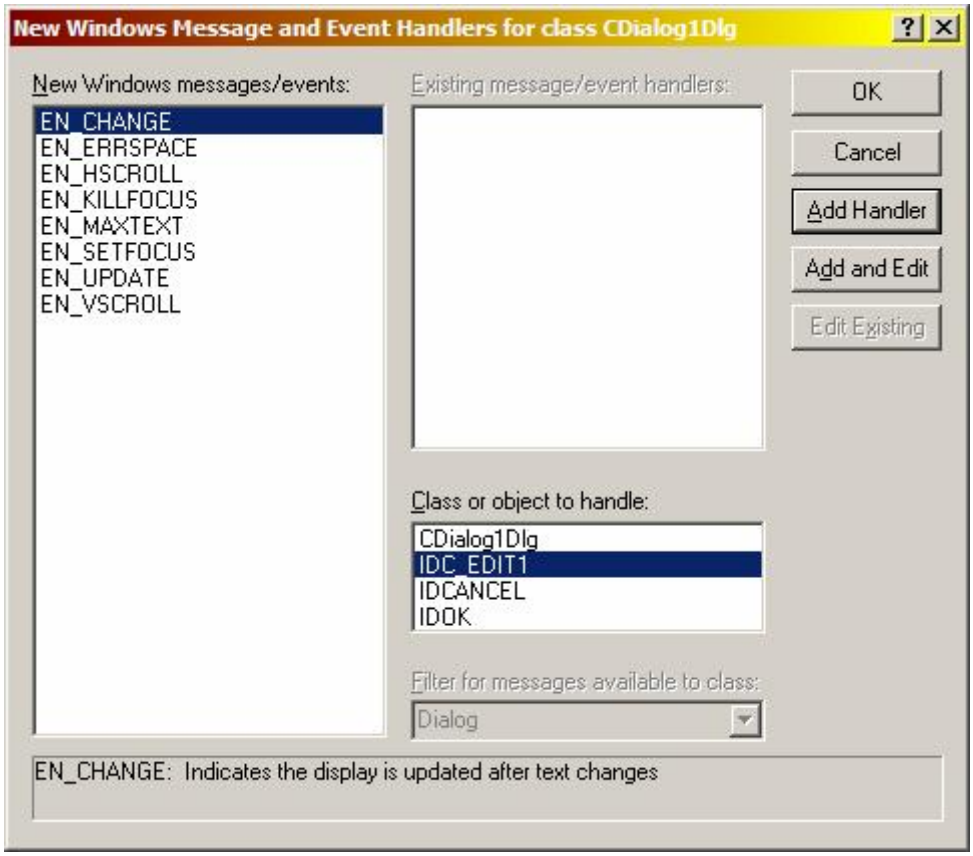
Jedes Control besitzt einige bestimmte Eigenschaften, die von den MFC-Klassen entsprechend gekapselt werden. Die verschiedenen Controls erzeugen auch jeweils verschiedene Benachrichtigungen, die wir im Dialog abfangen können. Diese Nachrichten sind so genannte Control-Notifications und haben häufig eine Bezeichnung dieser Art XN_BLAH. Das X steht für den Control-Typen (E für Edit, B für Button, usw.). Das N steht für Notification und das BLAH für die eigentliche Nachricht, die gesendet wird.

### CEdit

Das Edit-Feld wird durch ein Objekt der Klasse CEdit gekapselt. Die wichtigste Methode ist SetWIndowText um den Text zu setzen und GetWindowText um den Text zu lesen. Die Methode ist übrigens eine Methode der Basisklasse CWnd, von der jede Control-Klasse abgeleitet ist, denn schlussendlich ist jedes Control ein Fenster. Jedes Fenster hat einen Window-Text. Bei Fenstern mit Rahmen wird dieser als Titel verwendet bei Buttons ist das einfach der Text, der auf dem Knopf erscheint.
Das CEdit Control erzeugt eine Benachrichtigung, wenn der Text ändert. Diese Nachricht heisst EN_CHANGE.

### Einfügen eines Nachrichtenbehandlers

Durch Rechtklick auf die Dialogklasse können wir einen Nachrichten-Behandler vom Assistenten einfügen lassen.




Dialog zum Einfügen einer Nachrichtenbehandlung

Was die Nachrichten jeweils bedeuten findet man am besten mit der Hilfe von Microsoft heraus.

**CButton**

Diese Klasse bietet die Möglichkeit einen Button zu setzen oder in einer Check Box den angewählt Status zu setzen oder zu erfragen. Die Methoden heissen hierfür: SetState, SetCheck, GetState, GetCheck. Das wichtige Ereignis, das ein Button erzeugen kann ist: BN_CLICKED

**CCombobox**

Die Combobox bietet eine Methode AddString an um Einträge in Form eines Textes hinzuzufügen. Weitere Methoden ermöglichen es einen bestimmten Eintrag auszuwählen, oder den gewählten Eintrag herauszufinden. Die Combobox schickt unter anderem den Notification-Code CBN_SELCHANGE, wenn der Benutzer die Auswahl geändert hat.

**CListbox**

Auch diese Klasse bietet eine Methode AddString an. Überhaupt ist eine Listbox ähnlich wie eine Combobox, dient sie doch auch zur Auswahl aus einer Menge von verschiedenen Einträgen. Hier heisst die Nachricht LBN_SELCHANGE.

**CScrollbar**

Diese Klasse wird so häufig für Hauptfenster und derartiges verwendet, dass ihre Nachrichten nicht als Control-Nachrichten zu erkennen sind. Sie senden je nach dem die Windows-Nachricht WM_HSCROLL falls der Scrollbar horizontal liegt oder WM_VSCROLL, falls der Scrollbar vertikal steht.

**CSliderCtrl**

Mit Methoden zum setzen der möglichen Werte und zum Lesen der aktuellen Position ist dieses Control praktisch um den Benutzer einen Wert auswählen zu lassen, der nicht ganz genau bestimmt werden muss. Ein Beispiel ist die Lautstärke von irgendwas. Die Methoden heissen hierfür SetRange, SetPos, GetPos.

## Übung

**Dialog erstellen**

Stelle einen Dialog fertig, der diese Controls beinhaltet.

**Datenelemente einfügen**

Erzeuge Datenelemente für die verschiedenen Controls in der Dialog-Klasse mit Hilfe des Klassenassistenten.

**Experimentieren mit Methoden**

In der OnInitDialog Methode kannst du mit den Methoden auf den verschiedenen Datenelementen herumexperimentieren.

**Nachrichten behandeln**

Füge für die verschiedenen Nachrichten Behandlungsroutinen in den Dialog ein. Verwende hierfür auch den Assistenten.

# Ein individuelles Button Control

## Objektorientierung bei Controls

**Ausgangslage**

Eine Idee von Objektorientierung ist, dass Objekte sich möglichst um sich selber kümmern. Wenn wir aber nun Nachrichten behandeln lassen, landen die Methoden in der Dialogklasse, die diese Controls umgibt. Häufig ist das erwünscht, denn eine Änderung in einem Control kann zu einer Änderung in einem anderen Control führen.

**Idee**

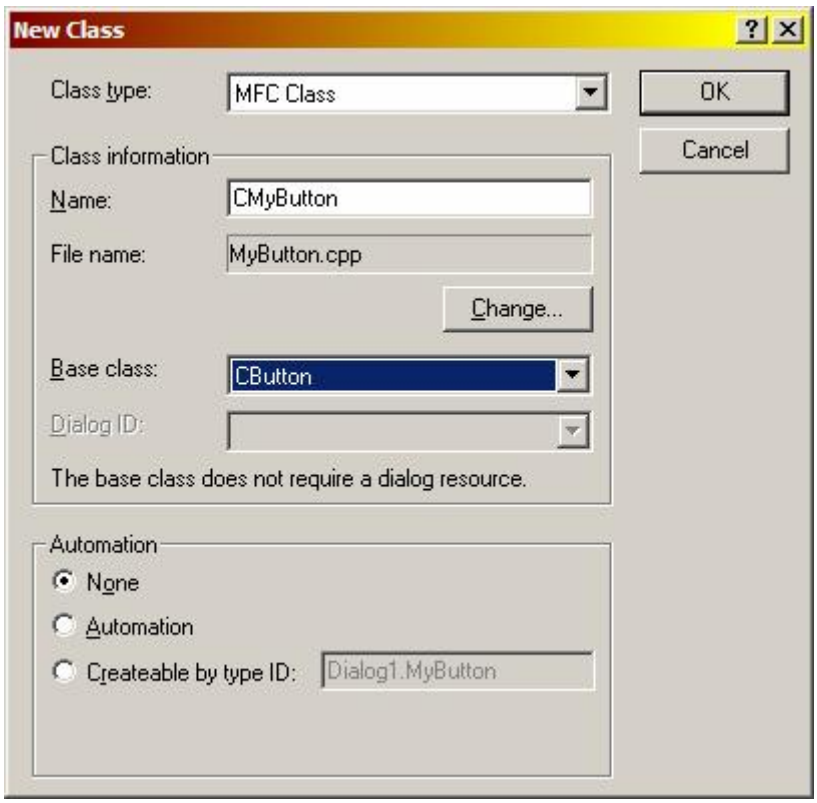
Das Framework lässt aber ohne weiteres zu, dass wir die MFC-Control Klassen erweitern und individuell gestalten. Um zu zeigen, was das bedeuten kann, versuchen wir eine Button-Klasse zu erzeugen, die sich

selber um ihre Darstellung kümmert und dadurch anders aussehen kann. In einer solchen Klasse können wir beliebige Nachrichten selber bearbeiten.

## Die Eigenen Button-Klasse

### Neue Klasse erzeugen

Erzeuge nun mit dem Assistenten eine neue Klasse durch Rechtsklick auf das Projekt in der Klassenansicht.




Dialog für eine neue Klasse

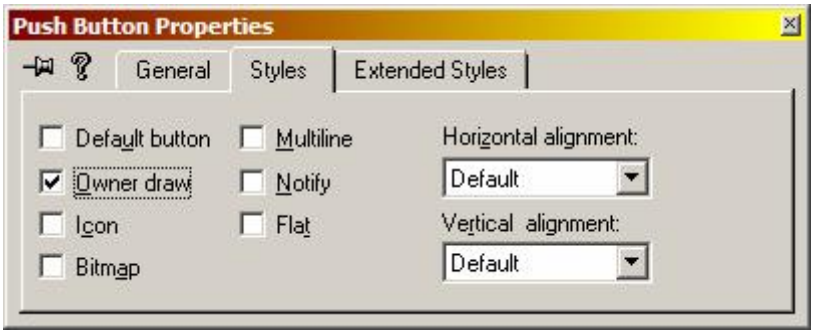
Wähle als Basisklasse die Klasse CButton aus.

### Verbinden eines Elementes mit unserer Klasse

Genauso wie wir vorhin ein Button-Control mit einem Element der Klasse CButton verbunden haben, können wir das Element nun mit einem Element der Klasse CMyButton verbinden. Die Auswahl sollte im Dialog zur Verfügung stehen. Danach reicht es in der Headerdatei der Dialogklasse ein #include für die neu erzeugte Klasse einzufügen. Nun ist das Objekt, dass sich auf dem Dialog befindet ein Objekt der Klasse CMyButton.

### Selber malen von Controls

Viele Controls bieten als Eigenschaft OWNERDRAW an. Das bedeutet, dass der Programmierer das Control selber zeichnet. Im Dialogeditor kann in den Eigenschaften des Buttons dieses Flag gesetzt werden.




Setzen der Ownderdraw-Eigenschaft

Nach dieser Änderung funktioniert unser Programm nicht mehr. Nach dem Start sollte es sofort in einen ASSERT reinlaufen:

```
void CButton::DrawItem(LPDRAWITEMSTRUCT)
{
    ASSERT(FALSE);
}
```

Über diesem Code steht, dass die abgeleitete Klasse verantwortlich ist diese Methode zu überschreiben.

**Die DrawItem Methode**

Also überschreiben wir nun in unserer Klasse CMyButton die virtuelle Methode DrawItem.

```
void CMyButton::DrawItem(LPDRAWITEMSTRUCT drawitem)
{
}
```

So macht unsere Methode nichts, aber das Programm läuft wieder. Nur unser Button ist verschwunden!
Kein Wunder, denn jetzt sind wir dafür verantwortlich unseren Button abhängig von seinem Zustand korrekt darzustellen.
In der Struktur, die unserer Methode übergeben wird, bekommen wir auch ein wenig Information hierzu. In dieser Struktur erhalten wir eine RECT Struktur, die dem Rechteck unseres Buttons entspricht. Ebenso erhalten wir einen HDC, den wir verwenden können um Bildschirmausgaben zu machen.

```
void CMyButton::DrawItem(LPDRAWITEMSTRUCT drawItem)
{
    // wir hole uns einen Zeiger auf ein DC Objekt
    CDC* dc = CDC::FromHandle(drawItem->hDC); //

    dc->TextOut(drawItem->rcItem.left,
                drawItem->rcItem.top,
                „Button“);
}
```

## Übung Eigener Button

Mit dem Wissen, das du bis hierhin angesammelt hast, sollte es möglich sein einen besonderen Button zu erzeugen.
Dieser Button, kann als Eigenschaft haben, dass er wenn er wenn er gedrückt wird (BN_CLICKED behandeln) seinen Zustand wechselt (SetCheck, GetCheck). In der DrawItem Methode kannst du diesen Zustand verwenden um den Button speziell darzustellen.
Du kannst sogar zwei Bitmaps zeichnen und versuchen diese darzustellen.