Steuerelemente

In nahezu jeder Windows-Anwendung begegnet man Schaltflächen, Kontrollkästchen, Textfeldern und DropDown-Listenfeldern. Man bezeichnet diese Komponenten als Steuerelemente und viele dieser Steuerelemente sind in das Betriebssystem selbst integriert. In Visual C++ lassen sich diese allgemeinen Steuerelemente einsetzen, indem man sie einfach per Drag&Drop in einem Dialogfeld platziert.

Am heutigen Tag lernen Sie, …

- welche grundlegenden Steuerelemente in Visual C++ zur Verfügung stehen,
- wie man Variablen deklariert und sie mit einem Steuerelement verbindet,
- wie man die Werte von Steuerelementen und Variablen synchronisiert,
- wie man die Reihenfolge festlegt, in der die Benutzer durch die Steuerelemente im Anwendungsfenster navigieren,
- wie man Aktionen mit Steuerelementen auslöst,
- wie man (bei laufender Anwendung) das Aussehen von Steuerelementen manipuliert und verändert.

3.1 Standardsteuerelemente von Windows

Zum Betriebssystem Windows gehören verschiedene Standardsteuerelemente wie Schieberegler, Strukturansicht, Listenelement oder Statusanzeige. In der heutigen Lektion geht es zunächst einmal um ein halbes Dutzend Steuerelemente, die sich in fast jeder Windows-Anwendung finden:

- Statisches Textfeld (Static Text)
- Eingabefeld für Text (Edit Control)
- Schaltfläche (Button)
- Kontrollkästchen (Check Box)
- Optionsfeld (Radio Button)
- Kombinationsfeld, auch als DropDown-Listenfeld bezeichnet (Combo Box, Drop Down List)

Diese und andere Steuerelemente stehen zur sofortigen Nutzung in Visual C++-Anwendungen bereit. Sie sind in der Toolbox-Ansicht im Dialogeditor von Developer Studio untergebracht (siehe Abbildung 3.1).

Abbildung 3.1:
Die in der Toolbox-Ansicht verfügbaren Standardsteuerelemente

Statischer Text

Das Steuerelement TEXT verwendet man, um Text für den Benutzer anzuzeigen. Der Benutzer kann diesen Text weder ändern noch in anderer Form mit dem Steuerelement interagieren. Das Steuerelement ist als Nur-Lesen-Element vorgesehen. Allerdings kann man im Code der laufenden Anwendung den durch das Steuerelement angezeigten Text ohne weiteres ändern.

Die wichtigsten Eigenschaften des Steuerelements TEXT sind in Tabelle 3.1 aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Der Standardwert ist immer IDC_STATIC. Wenn Sie das Steuerelement TEXT kontrollieren wollen, um sein Erscheinungsbild oder seinen TEXT während der Laufzeit zu ändern, müssen Sie die Eigenschaft ID ändern.

Tabelle 3.1: Wichtige Eigenschaften des Steuerelements Text

Eigenschaft	Beschreibung
Beschriftung	Legt den im Steuerelement TEXT angezeigten Text fest
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, während die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist. Wenn das Steuerelement deaktiviert ist, erscheint es grau hinterlegt. Der Beschriftungstext erscheint als Umriss oder Gravierung in der Oberfläche des Dialogfensters.
Tabstopp	Gibt an, ob Benutzer das Steuerelement durch Navigieren mit der Tabulatortaste erreichen können. Man könnte annehmen, dass diese Eigenschaft beim Steuerelement TEXT nichts zu suchen hat. Wenn Sie dem Beschriftungstext allerdings Kurztasten zuweisen, wird diese Eigenschaft äußerst wichtig, da der Fokus dem nächsten Steuerelement nach dem Text zugewiesen wird.

Tabelle 3.1: Wichtige Eigenschaften des Steuerelements Text (Forts.)

MFC-Exkurs: Die Klasse `CStatic`

Für jedes Steuerelement, das in einer Windows-Anwendung benutzt werden kann, existiert eine entsprechende MFC-Klasse. Diese Klasse verkapselt die gesamte Steuerfunktionalität und bietet damit eine einfache Möglichkeit, mit dem Steuerelement zu interagieren. Das Steuerelement Text ist in der Klasse `CStatic` verkapselt.

Die Klasse `CStatic` ist ein Abkömmling der Klasse `CWnd`, der Basis-Klasse für alle visuellen Komponenten in einer MFC-Anwendung. Die Klasse `CStatic` wird gewöhnlich für die Anzeige von TEXT verwendet, mit dem der Benutzer nicht interagieren kann.

Eingabefelder

In einem Eingabefeld kann der (spätere) Anwender Text eingeben oder ändern. Das Steuerelement gehört zu den Hauptwerkzeugen, die dem Benutzer die Möglichkeit bieten, einer Anwendung bestimmte erforderliche Informationen bereitzustellen. Es ist in der Lage, beliebigen Text bis zu einer bestimmten Länge aufzunehmen, der sich auslesen und nach Bedarf weiterverarbeiten lässt. Das Eingabefeld akzeptiert ausschließlich reinen Text, Formatierungen stehen dem Benutzer nicht zur Verfügung. Wichtige Eigenschaften von Eingabefeldern sind in Tabelle 3.2 aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Sie müssen diese Eigenschaft ändern, um das Eingabefeld identifizieren und mit ihm arbeiten zu können.
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, wenn die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist
Text ausrichten (Align Text)	Gibt an, ob der Text im Steuerelement am rechten oder linken Rand ausgerichtet oder zentriert ist
Mehrfachzeile (Multiline)	Gibt an, ob sich der Text im Eingabefeld über mehrere Zeilen erstrecken kann
Zahl	Beschränkt den Text, den ein Eingabefeld annimmt, auf Zahlen
Kennwort	Verbirgt den Text im Eingabefeld und zeigt stattdessen Sterne (`'*'`)
Tabstopp	Gibt an, ob Benutzer das Steuerelement beim Navigieren mit der Tabulatortaste erreichen können

Tabelle 3.2: Wichtige Eigenschaften von Eingabefeldern

MFC-Hinweis: Die Klasse `CEdit`

Das Steuerelement EINGABEFELD ist in der Klasse `CEdit` verkapselt. Die Klasse `CEdit` ist ein Abkömmling der Klasse `CWnd` und erweitert diese durch verschiedene Methoden für die Bearbeitung von und die Interaktion mit Text im Steuerelement. Die Klasse `CEdit` enthält sogar Methoden für die Interaktion mit der Zwischenablage, darunter `Copy` (Kopieren), `Cut` (Ausschneiden), `Paste` (Einfügen) und `Undo` (Rückgängig), also die Methoden, die keine Parameter übernehmen.

Schaltflächen

Über eine Schaltfläche löst der Benutzer eine bestimmte Aktion aus. Die Beschriftung oder Caption – der Titel – der Schaltfläche sollte einen Hinweis auf die Aktion liefern, die beim Klicken auf die Schaltfläche ausgeführt wird. Eine Schaltfläche kann auch Bilder enthalten, die man – allein oder zusammen mit einer Textbeschreibung – benutzt, um den Zweck der Schaltfläche zu vermitteln. Wichtige Eigenschaften des Steuerelements SCHALTFLÄCHE sind in Tabelle 3.3 aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Sie müssen diese Eigenschaft ändern, um die Schaltfläche identifizieren und mit ihr interagieren zu können.
Beschriftung	Gibt den auf der Schaltfläche angezeigten Text an
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, wenn die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist. Der Text auf der Schaltfläche erscheint als Umriss oder Gravierung in der Oberfläche des Dialogfensters.
Standardschaltfläche (Default Button)	Gibt an, dass dieses Steuerelement ausgelöst werden soll, wenn der Benutzer die Eingabetaste betätigt
Tabstopp	Gibt an, ob Benutzer das Steuerelement beim Navigieren mit der Tabulatortaste erreichen können

Tabelle 3.3: Wichtige Eigenschaften des Steuerelements Schaltfläche

MFC-Hinweis: Die Klasse `CButton`

Die Klasse `CButton` ist die MFC-Klasse, die das Steuerelement SCHALTFLÄCHE verkapselt. Diese Klasse ist ein Abkömmling der Klasse `CWnd` und verkapselt nicht nur die Schaltfläche, sondern auch das Kontrollkästchen und das Optionsfeld. Sie wird von der Klasse `CBitmapButton` ererbt, mit der Sie ein Bild auf der Schaltfläche darstellen können.

Sie können den Zustand einer Schaltfläche mit den Methoden `GetState` und `SetState` abfragen und ändern. Sie können das Erscheinungsbild und die Verhaltensweise der Schaltfläche mit den Methoden `GetButtonStyle` und `SetButtonStyle` kontrollieren.

Kontrollkästchen

Als Kontrollkästchen bezeichnet man die kleinen quadratischen Elemente, die der Benutzer durch Anklicken ein- bzw. ausschaltet und damit einen bestimmten Wert setzt bzw. zurücksetzt. Grundsätzlich handelt es sich um An-/Ausschalter, gelegentlich mit einem dritten Zwischenzustand. In Tabelle 3.4 sind wichtige Eigenschaften des Steuerelements KONTROLLKÄSTCHEN aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Sie müssen diese Eigenschaft ändern, um das Kontrollkästchen identifizieren und mit ihm interagieren zu können.
Beschriftung	Gibt den Text an, der im Steuerelement angezeigt wird
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, wenn die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist. Der Beschriftungstext erscheint als Umriss oder Gravierung in der Oberfläche des Dialogfensters.
Drei-Status (Tri-state)	Gibt an, dass das Kontrollkästchen drei Zustände anstatt der gewöhnlichen zwei annehmen kann. Der dritte Zustand ist Deaktiviert, das heißt der Wert des Steuerelements ist weder TRUE noch FALSE.
Tabstopp	Gibt an, ob Benutzer das Steuerelement beim Navigieren mit der Tabulatortaste erreichen können

Tabelle 3.4: Wichtige Eigenschaften des Steuerelements Kontrollkästchen

Optionsfelder

Ein Optionsfeld wird als Kreis dargestellt. Klickt der Benutzer ein Optionsfeld an, erscheint ein Punkt im Kreis. Das Optionsfeld ähnelt dem Kontrollkästchen, wird aber in einer Gruppe verwendet, in der nur ein Optionsfeld eingeschaltet sein kann. Normalerweise setzt man Optionsfelder in Gruppen mit mindestens zwei Optionen ein, wobei die Optionsfelder von einem Gruppenfeld umgeben sind. Das Gruppenfeld gewährleistet die visuelle Unabhängigkeit der einzelnen Optionsfeldgruppen und weist den Benutzer darauf hin, dass in jeder Gruppe jeweils nur ein Optionsfeld eingeschaltet sein kann. In Tabelle 3.5 sind wichtige Eigenschaften der Optionsfelder aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Sie müssen diese Eigenschaft ändern, um das Optionsfeld identifizieren und mit ihm interagieren zu können.
Beschriftung	Gibt den Text an, der im Steuerelement angezeigt wird
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, wenn die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist. Der Beschriftungstext erscheint als Umriss oder Gravierung in der Oberfläche des Dialogfensters.

Tabelle 3.5: Wichtige Eigenschaften des Steuerelements Optionsfeld

Eigenschaft	Beschreibung
Gruppe (Group)	Gibt an, dass ein Steuerelement das erste in einer Gruppe von Steuerelementen ist. Diese Eigenschaft ist für die meisten Steuerelemente verfügbar, für Optionsfelder ist sie jedoch besonders wichtig. Diese Eigenschaft sollte nur beim ersten Optionsfeld in einer Gruppe von Optionsfeldern auf TRUE gesetzt werden.
Auto	Veranlasst das Optionsfeld, automatisch seinen Status zu ändern, wenn es angeklickt wird. Standardmäßig ist diese Eigenschaft auf TRUE gesetzt.
LeftText	Veranlasst die Beschriftung, links vom Optionsfeld zu erscheinen. Normalerweise befindet sie sich rechts davon.
Tabstopp	Gibt an, ob Benutzer das Steuerelement beim Navigieren mit der Tabulatortaste erreichen können

Tabelle 3.5: Wichtige Eigenschaften des Steuerelements Optionsfeld (Forts.)

Kombinationsfelder

Ein Kombinationsfeld (oder DropDown-Listenfeld) besteht aus einem Eingabefeld, dem eine Liste mit verfügbaren Werten zugeordnet ist. Mit einem Kombinationsfeld kann man eine Liste von Auswahlfeldern bereitstellen, wobei der Benutzer einen Wert aus der Liste auswählen kann. Manchmal hat der Benutzer die Möglichkeit, selbst einen Wert einzutippen, wenn die Liste keinen passenden Wert enthält. Wichtige Eigenschaften für Kombinationsfelder sind in Tabelle 3.6 aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Sie müssen diese Eigenschaft ändern, um das Kombinationsfeld identifizieren und mit ihm interagieren zu können.
Beschriftung	Gibt den Text an, der im Steuerelement angezeigt wird
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, wenn die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist. Der Beschriftungstext erscheint als Umriss oder Gravierung in der Oberfläche des Dialogfensters.
Sortieren (Sortierenieren)	Kontrolliert, ob die Einträge in der DropDown-Liste sortiert sind

Tabelle 3.6: Wichtige Eigenschaften von Kombinationsfeldern

Eigenschaft	Beschreibung
Typ (Type)	Gibt an, welche Art Kombinationsfeld anzuzeigen ist: »*Einfach*«, »*DropDown*« (der Standard) oder »*DropDown-Listenfeld*«. »*Einfach*« zeigt die Liste immer an. im »*DropDown*«-Stil wird die Liste nur angezeigt, wenn das Steuerelement den Fokus hat, entweder weil der Benutzer es mit der Tabulatortaste angesprungen oder weil er auf den Pfeil geklickt hat, um die Liste zu öffnen. Bei »*DropDown-Listenfeld*« wird die aktuelle Auswahl in einem Text-Steuerelement angezeigt; ansonsten entspricht dieser Typ dem Typ »*DropDown*«.
Tabstopp	Gibt an, ob Benutzer das Steuerelement beim Navigieren mit der Tabulatortaste erreichen können

Tabelle 3.6: Wichtige Eigenschaften von Kombinationsfeldern (Forts.)

MFC-Hinweis: Die Klasse `CComboBox`

Die Klasse `CComboBox` verkapselt das Steuerelement KOMBINATIONSFELD. Sie ist ein Abkömmling der Klasse `CWnd` und wird von der Klasse `CComboBoxEx` ererbt. Die Klasse `CComboBoxEx` stellt Funktionalität bereit, um Bilder in die Auswahl der DropDown-Liste aufzunehmen.

Die Klasse `CComboBox` stellt Funktionalität für die Interaktion mit der DropDown-Liste bereit. Sie bietet Funktionen, um die aktuelle Auswahl mit den Methoden `GetCurSel` und `SetCurSel` sowie `SelectString` und `FindString` abzufragen und zu setzen. Sie ermöglicht Ihnen auch, mit den Methoden `AddString`, `InsertString` und `DeleteString` Elemente in die Liste einzufügen und daraus zu entfernen. Wenn Sie den Inhalt der Liste zurücksetzen und neu beginnen müssen, können Sie die Methode `ResetContent` verwenden.

3.2 Steuerelemente in ein Fenster aufnehmen

Die heute zu erstellende Anwendung soll eine Reihe von Steuerelementen in einem Dialogfeld enthalten, wie es aus Abbildung 3.2 hervorgeht. Diese Steuerelemente haben verschiedene Aufgaben.

- Am oberen Rand des Fensters befindet sich ein Eingabefeld, in das der Benutzer eine Nachricht eintippen kann, die in einem Meldungsfeld erscheint, wenn er auf die Schaltfläche neben dem Feld klickt.
- Unterhalb dieses Eingabefelds sind zwei Schaltflächen angeordnet, die entweder das Eingabefeld mit einer Standardmeldung füllen oder einen vorhandenen Eintrag löschen.

- Unter diesen Schaltflächen befindet sich ein Kombinationsfeld mit einer Liste von Windows-Standardanwendungen. Wählt der Benutzer eines dieser Programme aus und klickt auf die Schaltfläche neben der DropDown-Liste, startet das ausgewählte Programm.
- Unterhalb des Kombinationsfelds sind zwei Gruppen von Kontrollkästchen angeordnet. Diese wirken auf die Steuerelemente im oberen Teil des Dialogfelds: die Steuerelemente für die Anzeige einer Benutzermeldung und die Steuerelemente für die Ausführung eines anderen Programms.
- Die Kontrollkästchen auf der linken Seite aktivieren und deaktivieren die einzelnen Gruppen von Steuerelementen.
- Mit den rechten Kontrollkästchen lassen sich die Gruppen der Steuerelemente anzeigen und ausblenden.
- Im unteren Teil des Dialogfelds befindet sich eine Schaltfläche, die das Schließen der Anwendung bewirkt.

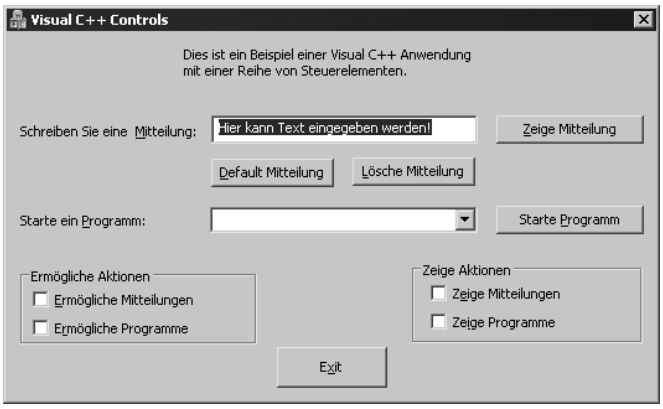

Abbildung 3.2:
Die heutige Anwendung verwendet mehrere Standardsteuerelemente.

Anwendungsgerüst und Layout des Dialogfelds erstellen

Mit den gestern erworbenen Kenntnissen können Sie jetzt ein neues Anwendungsgerüst erstellen und das Layout des Anwendungsdialogfelds entwerfen. Führen Sie dazu die folgenden Schritte aus:

1. Erstellen Sie ein neues MFC-Application-Visual C++-Projekt und nennen Sie es `Controls`.
2. Verwenden Sie im MFC Anwendungs-Assistent die gleichen Einstellungen wie an den letzten beiden Tagen. Legen Sie den Titel des Dialogfelds mit `Visual C++ Controls` fest.

3. Nachdem Sie das Anwendungsgerüst erstellt haben, gestalten Sie das Hauptdialogfeld wie in der weiter oben gezeigten Abbildung 3.2.
4. Konfigurieren Sie die Eigenschaften der Steuerelemente gemäß Tabelle 3.7.

Steuerelement	Eigenschaft	Einstellung
Text	Beschriftung	`Dies ist ein Beispiel einer Visual C++ Anwendung mit einer Reihe von Steuerelementen.`
Text	ID Beschriftung	`IDC_STATICMSG` `Schreiben Sie eine &Mitteilung:`
Text	ID Beschriftung	`IDC_STATICPGM` `Starte ein &Programm:`
Eingabefeld	ID	`IDC_MSG`
Schaltfläche	ID Beschriftung	`IDC_SHWMSG` `&Zeige Mitteilung`
Schaltfläche	ID Beschriftung	`IDC_DFLTMSG` `&Default Mitteilung`
Schaltfläche	ID Beschriftung	`IDC_CLRMSG` `&Lösche Mitteilung`
Schaltfläche	ID Beschriftung	`IDC_RUNPGM` `Starte &Programm`
Schaltfläche	ID Beschriftung	`IDC_EXIT` `E&xit`
Kombinationsfeld	ID	`IDC_PROGTORUN`
Gruppenfeld	Beschriftung	`Ermögliche Aktionen`
Gruppenfeld	Beschriftung	`Zeige Aktionen`
Kontrollkästchen	ID Beschriftung	`IDC_CKENBLMSG` `&Ermögliche Mitteilungen`
Kontrollkästchen	ID Beschriftung	`IDC_CKENBLPGM` `E&rmögliche Programme`
Kontrollkästchen	ID Beschriftung	`IDC_CKSHWMSG` `Z&eige Mitteilungen`
Kontrollkästchen	ID Beschriftung	`IDC_CKSHWPGM` `Ze&ige Programme`

Tabelle 3.7: Eigenschaftseinstellungen für die Steuerelemente im Anwendungsdialogfeld

5. Nachdem Sie alle genannten Steuerelemente im Dialogfeld platziert und deren Eigenschaften konfiguriert haben, markieren Sie das Kombinationsfeld noch einmal und suchen Sie im Eigenschaftenbereich die Eigenschaft DATEN. Geben Sie die folgenden Werte getrennt durch Semikola ein (siehe Abbildung 3.3).
   - Notepad
   - Paint
   - Solitaire

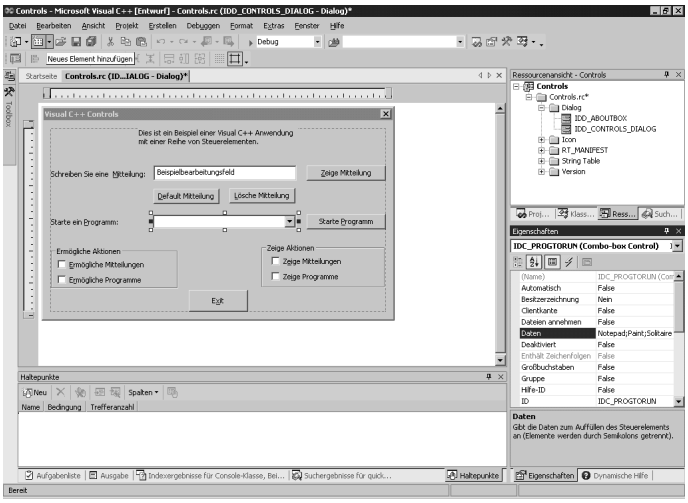

Abbildung 3.3: Die Einträge in der Drop-Down-Liste des Kombinationsfelds legen Sie in der Eigenschaftenansicht fest.

Wenn man ein Kombinationsfeld in das Fenster aufnimmt, ist darauf zu achten, dass man klickt und den Bereich für das Steuerelement so groß zieht, wie die DropDown-Liste sein soll. Nachdem man das Steuerelement im Fenster gezeichnet hat, kann man die Größe genauso ändern, wie es normalerweise üblich ist. Um die Ausdehnung der Liste nach unten festzulegen, klickt man auf den Pfeil – genauso, als würde man bei laufender Anwendung die Drop-Down-Liste aktivieren.

Die Tabulator-Reihenfolge von Steuerelementen festlegen

Nach der Anordnung der Steuerelemente im Fenster müssen Sie noch sicherstellen, dass der Benutzer bei der Navigation mithilfe der ⇆-Taste die Steuerelemente in der von Ihnen gewünschten Reihenfolge anspricht. Die Tabulator-Reihenfolge legen Sie mit den nachstehenden Schritten fest:

1. Markieren Sie im Bearbeitungsbereich von Visual Studio entweder das Dialogfeld oder eines der Steuerelemente im Fenster.
2. Wählen Sie FORMAT / TABULATOR-REIHENFOLGE. Daraufhin erscheinen im Fenster neben den Steuerelementen Nummern. Diese kennzeichnen die Reihenfolge, in der die Navigation durch das Dialogfeld verläuft (siehe Abbildung 3.4).

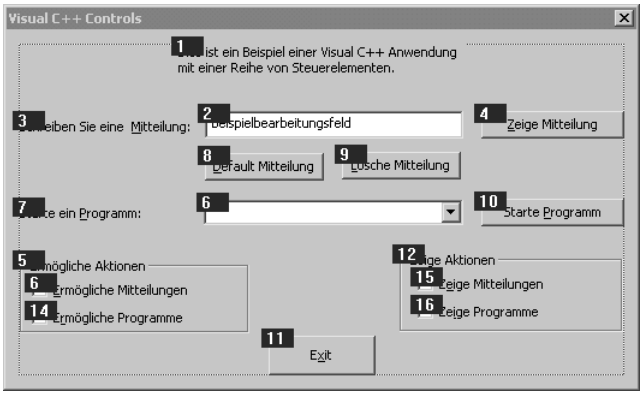

Abbildung 3.4:
Das Einschalten der Tabulator-Reihenfolge zeigt die Reihenfolge der Navigation durch das Dialogfeld an.

   Klicken Sie mit der Maus die Nummernfelder in der Reihenfolge an, in der der Benutzer durch das Dialogfeld navigieren soll. Die Steuerelemente nummerieren sich automatisch neu, wenn Sie sie nacheinander auswählen.

3. Nachdem Sie die Tabulator-Reihenfolge festgelegt haben, wählen Sie FORMAT / TABULATOR-REIHENFOLGE erneut, um zum Dialog-Editor zurückzukehren.

Statischer Text, der mit einer Zugriffstaste versehen ist, sollte in der Tabulator-Reihenfolge unmittelbar vor dem zugehörigen Steuerelement stehen. Da der Benutzer nicht mit statischem Text interagieren kann, geht der Fokus bei Wahl der Zugriffstaste direkt zum nächsten Steuerelement laut Tabulator-Reihenfolge.

Eine Zugriffstaste ist durch das unterstrichene Zeichen – den so genannten mnemonischen Code – in der Beschriftung einer Schaltfläche, eines Kontrollkästchens, eines Menüs oder eines anderen Steuerelements gekennzeichnet. Der Benutzer kann den unterstrichenen Buchstaben in Kombination mit der Alt-Taste drücken, um direkt zu diesem Steuerelement zu gelangen oder das angeklickte Ereignis auf dem Steuerelement auszulösen. Um eine Zugriffstaste festzulegen, schreibt man bei Eingabe des Titels ein kaufmännisches Und-Zeichen (&) unmittelbar vor das betreffende Zeichen. Achten Sie darauf, dass Sie für mehrere Zugriffstasten nicht dieselben Zeichen in demselben Fenster oder derselben Menügruppe vorsehen, da es den Benutzer nur verwirrt, wenn er eine Zugriffstaste drückt und nicht die erwartete Reaktion eintritt.

Bevor Sie sich mit dem Code der Anwendung beschäftigen, sollten Sie abschließend die Zugriffstasten auf Konflikte prüfen. Führen Sie dazu die folgenden Schritte aus:

1. Markieren Sie im Dialog-Editor das Dialogfeld oder eines der Steuerelemente. Klicken Sie mit der rechten Maustaste und wählen Sie MNEMONIK ÜBERPRÜFEN.
2. Wenn keine Konflikte bei Ihren mnemonischen Codes aufgetreten sind, zeigt Visual C++ ein entsprechendes Meldungsfeld an (siehe Abbildung 3.5).

Abbildung 3.5:
Eine Prüfung der Zugriffstasten zeigt, ob Konflikte vorliegen.

3. Falls Konflikte vorhanden sind, meldet das Dialogfeld den betreffenden Buchstaben und gibt Ihnen die Möglichkeit, die Steuerelemente mit den gegensätzlichen Einträgen auswählen zu lassen (siehe Abbildung 3.6).

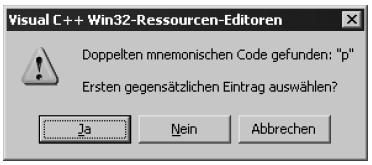

Abbildung 3.6:
Doppelte mnemonische Codes lassen sich automatisch auswählen.

4. In den Beschreibungen von Tabelle 3.7 waren zwei doppelte mnemonische Codes angegeben, wählen Sie passenden andere Buchstabenkombinationen.

3.3 Variablen mit Steuerelementen verbinden

Wenn Sie bereits mit Visual Basic oder PowerBuilder programmiert haben, ahnen Sie sicherlich schon, dass es nun an der Zeit ist, etwas Code zu verfassen. Bei Visual C++-Anwendungen, die mit MFC erstellt werden, läuft dieser Prozess allerdings nicht genauso ab. Bevor Sie mit der Codierung beginnen können, müssen Sie allen Steuerelementen, denen ein Wert zugewiesen ist, Variablen zuordnen – allen, außer dem statischem Text und den Schaltflächen. Auf die Variablen greifen Sie zurück, wenn Sie den Code für die Anwendung schreiben. Die Werte, die der Benutzer in die Bildschirmsteuerelemente eingibt, übernimmt das Programm zur Weiterverarbeitung in diese Variablen. Analog dazu werden alle Werte, die der Code Ihrer Anwendung in diese Variablen stellt, in den Steuerelementen des Fensters aktualisiert, damit sie der Benutzer sieht.

Wie deklariert man nun diese Variablen und verbindet sie mit den Steuerelementen, die man im Fenster platziert hat? Führen Sie die folgenden Schritte aus:

1. Wählen Sie das Steuerelement aus, das Sie mit einer Variablen verbinden möchten.
2. Rechtsklicken Sie mit der Maus auf dem Steuerelement und wählen Sie aus dem erscheinenden Kontext-Menü VARIABLE HINZUFÜGEN.
3. Wählen Sie die ID von einem der Steuerelemente, mit dem Sie eine Variable verbinden möchten, wie beispielsweise `IDC_MSG`. Die ID des gewählten Steuerelements sollte im Kombinationsfeld STEUERELEMENT-ID bereits hervorgehoben sein.
4. Geben Sie VALUE (Wert) als KATEGORIE an.
5. Wählen Sie aus dem Kombinationsfeld VARIABLENTYP den Datentyp der Variablen aus, hier wird bereits vom Assistenten ein sinnvoller Vorschlag gemacht.
6. Geben Sie im Eingabefeld VARIABLENNAME einen Namen ein.
7. Wenn Sie möchten geben Sie im Eingabefeld KOMMENTAR am unteren Rand des Dialogs (siehe Abbildung 3.7) einen Kommentar ein, der die Variable und ihre Verwendung beschreibt. Klicken Sie auf FERTIG STELLEN, um die Variable hinzuzufügen.

Wiederholen Sie die Schritte 1-7 für alle anderen Steuerelemente, für die Sie Variablen zuordnen müssen. In der heutigen Anwendung betrifft das die Variablen gemäß Tabelle 3.8.

Steuerelement	Variablenname	Kategorie	Typ	Zugriff
`IDC_MSG`	`m_strMessage`	Value (Wert)	`CString`	`public`
`IDC_PROGTORUN`	`m_strProgToRun`	Value	`CString`	`public`
`IDC_CKENBLMSG`	`m_bEnableMsg`	Value	`BOOL`	`public`
`IDC_CKENBLPGM`	`m_bEnablePgm`	Value	`BOOL`	`public`
`IDC_CKSHWMSG`	`m_bShowMsg`	Value	`BOOL`	`public`
`IDC_CKSHWPGM`	`m_bShowPgm`	Value	`BOOL`	`public`

Tabelle 3.8: Variablen für die Steuerelemente der Anwendung

Wenn Sie einen Datentyp benötigen, der sich im Kombinationsfeld mit den Datentypen nicht findet, können Sie im Eingabebereich des Kombinationsfelds Ihren eigenen Datentyp angeben.

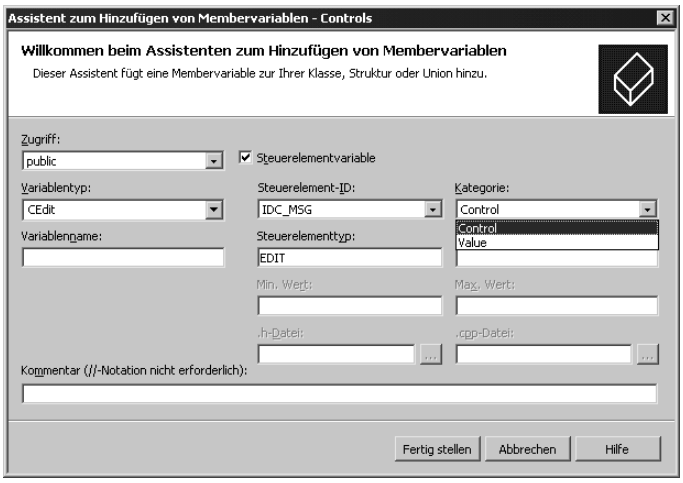

Abbildung 3.7: Einem Steuerelement eine Variable hinzufügen

Alle hier verwendeten Variablen beginnen mit dem Präfix `m_`, da es sich um Member-Variablen (oder Elementvariablen) einer Klasse handelt. Diese Konvention hat sich bei der Vergabe von Namen für Elemente von MFC-Klassen eingebürgert. Nach dem `m_` verwendet man eine Variante der Ungarischen Notation, bei der die nächsten Buchstaben den Variablentyp beschreiben. Im obigen Beispiel bedeutet `b` einen Booleschen Wert, während `str` eine Variable als String (Zeichenfolge) kennzeichnet. Diese Namenskonvention finden Sie im vorliegenden Buch und auch in anderen Büchern, die sich dem Thema Programmierung mit Visual C++ und MFC widmen. Durch diese Schreibweise lässt sich der Code für andere Programmierer leichter erfassen, was umgekehrt auch auf Sie zutrifft.

3.4 Steuerelemente mit Funktionalität ausstatten

Bevor Sie den Code für alle Steuerelemente in Ihrem Anwendungsfenster schreiben, ist zunächst etwas Code erforderlich, um die Variablen zu initialisieren, d.h. Startwerte für die meisten Variablen festzulegen. Führen Sie dazu folgende Schritte aus:

1. Erweitern Sie in der Klassen-Ansicht die Klasse `CControlDlgs` und wählen Sie die Funktion `OnInitDialog` aus der Liste der Member-Funktionen.
2. Doppelklicken Sie auf den Knoten von `OnInitDialog`, um zum Quellcode der Funktion zu gelangen.

Gehen Sie zur Markierung `TODO`, die die Stelle kennzeichnet, wo Sie mit der Codeeingabe beginnen. Geben Sie hier den Code aus Listing 3.1 ein.

Listing 3.1: In die Funktion OnInitDialog geben Sie den Initialisierungscode ein.

```
 1: BOOL CControlsDlg::OnInitDialog()
 2: {
 3:    CDialog::OnInitDialog();
 4: ...
 5:   // Symbol für dieses Dialogfeld festlegen. Wird automatisch
 6:   // erledigt
 7:   //  wenn das Hauptfenster der Anwendung kein Dialogfeld ist
 8:   SetIcon(m_hIcon, TRUE);        // Großes Symbol verwenden
 9:   SetIcon(m_hIcon, FALSE);       // Kleines Symbol verwenden
10:
11:   // TODO: Hier zusätzliche Initialisierung einfügen
12:   // Eine Standardmeldung in das Nachrichteneingabefeld setzen
13:   m_strMessage = "Schreiben Sie hier eine Mitteilung";
14:
15:   // Alle Kontrollkästchen auf aktiviert setzen
16:   m_bShowMsg = TRUE;
17:   m_bShowPgm = TRUE;
18:   m_bEnableMsg = TRUE;
19:   m_bEnablePgm = TRUE;
20:   // Den Dialog mit den Werten aktualisieren
21:   UpdateData(FALSE);
22:   return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement
24:                // soll den Fokus erhalten
25: }
```

Listing 3.1 zeigt nur einen Ausschnitt der Funktion `OnInitDialog`. Die Listings im gesamten Buch konzentrieren sich nur auf den Code, der hinzuzufügen oder zu modifizieren ist, und zeigen der Übersichtlichkeit wegen nicht den gesamten Code aller Funktionen. (Damit bleibt auch der Umfang des Buches in einem vernünftigen Rahmen.) Um Ihre Kenntnisse zu MFC und Visual C++ zu erweitern, sollten Sie sich auch den Code ansehen, der in den Listings im Buch ausgelassen wurde. Auf der Buch-CD finden Sie natürlich den vollständigen Code. Versuchen Sie zu verstehen, was der Code bewirkt.

Wenn Sie bereits in C oder C++ programmiert haben, ist Ihnen sicherlich aufgefallen, dass der Wert der Variablen `m_strMessage` in einer C-untypischen Art und Weise festgelegt wird. Das Ganze erinnert mehr an die Festlegung einer Stringvariablen in Visual Basic oder PowerBuilder. Das hängt damit zusammen, dass diese Variable vom Typ `CString` ist. Die Klasse `CString` erlaubt es, in

einer Visual C++-Anwendung mit Strings zu arbeiten, genau wie man mit Strings in einer der anderen Programmiersprachen umgeht. Da es sich jedoch um die Programmiersprache C++ handelt, müssen Sie dennoch ein Semikolon am Ende eines jeden Statements schreiben.

Der Initialisierungscode ist nicht weiter kompliziert. Das Programm stellt zuerst in das Eingabefeld eine anfängliche Nachricht, die Sie dem Benutzer anzeigen wollen, und setzt dann alle Kontrollkästchen in den eingeschalteten Zustand. Die letzte Zeile des hinzugefügten Codes verdient etwas mehr Beachtung.

Die Funktion `UpdateData` bildet den Schlüssel für die Arbeit mit Steuerelementvariablen in Visual C++. Die Funktion übernimmt die Daten aus den Variablen und aktualisiert mit den Werten die Steuerelemente auf dem Bildschirm. Umgekehrt übernimmt die Funktion die Daten aus den Steuerelementen und füllt die zugeordneten Variablen mit allen vom Benutzer geänderten Werten. Die Richtung der Datenübertragung steuert man mit dem an die Funktion `UpdateData` übergebenen Argument.

Ist das Argument auf `FALSE` gesetzt, werden die Werte in den Variablen an die Steuerelemente im Fenster übertragen. Übergibt man `TRUE` als Argument, erhalten die Variablen die aktuellen Werte der Steuerelemente im Fenster. Welcher Wert an die Funktion zu übergeben ist, hängt also davon ab, in welcher Richtung die Aktualisierung stattfinden soll.

Nachdem Sie eine oder mehrere Variablen in Ihrem Code aktualisiert haben, müssen Sie `UpdateData` aufrufen und `FALSE` als Argument übergeben. Wenn Sie die Variablen lesen müssen, um deren aktuellen Wert zu erhalten, ist `UpdateData` mit dem Argument `TRUE` aufzurufen, bevor Sie irgendeinen Wert aus den Variablen verarbeiten. Ein Gefühl für diese Vorgehensweise werden Sie entwickeln, wenn Sie mehr Code in Ihre Anwendung aufnehmen.

Achten Sie sorgfältig darauf, wo Sie die Funktion `UpdateData` aufrufen und welchen Wert Sie ihr übergeben. Wenn Sie beispielsweise die Werte all Ihrer Variablen initialisieren und dann `UpdateData(TRUE)` aufrufen, löschen Sie die Werte der Variablen aus und ersetzen sie durch die Werte der Steuerelemente. Wenn andererseits ein Benutzer den Wert in einem der Steuerelemente ändert und dann `UpdateData(FALSE)` aufgerufen wird, ändert sich der Wert des Steuerelements in den Wert der Variablen und die Eingabe des Benutzers wird gelöscht.

C++-Exkurs: Der Scope- Resolution-Operator

Wenn Sie den Codeausschnitt in Listing 3.1 betrachten, sehen Sie folgende Code-Zeile:

```
CDialog::OnInitDialog();
```

Basierend auf dem, was Sie bisher gesehen haben, sieht das wie die erste Zeile des Funktionslistings aus, wo die Klasse gefolgt von zwei Doppelpunkten und dem Funktionsnamen angegeben ist:

```
BOOL CControlsDlg::OnInitDialog()
```

In der ersten Zeile des Listings steht der Funktionsname mit dem Rückgabetyp und der Klasse, deren Mitglied die Funktion ist. Die dritte Zeile ruft dieselbe Funktion in der Basisklasse auf. So rufen Sie in C++ Basisklassen-Funktionen auf, indem Sie den Namen der Basisklasse gefolgt von zwei Doppelpunkten und dem Namen der Funktion angeben und alle benötigten Parameter übergeben.

Die beiden Doppelpunkte ohne Zwischenraum werden als *Scope-Resolution-Operator* bezeichnet und verwendet, um den Geltungsbereich (Scope) der Funktion anzugeben. In den beiden obigen Situationen gibt er an, dass die aufzurufende Funktion ein Mitglied der Klasse vor den beiden Doppelpunkten ist. Wenn Sie eine Funktion mit zwei Doppelpunkten vor dem Funktionsaufruf und ohne Klassennamen davor haben, gibt der Scope-Resolution-Operator an, dass es sich um eine globale Funktion handelt und nicht um ein Mitglied einer Klasse.

Sie können den Scope-Resolution-Operator auch verwenden, um den Geltungsbereich von Variablen zu definieren. Diese Verwendung findet sich allerdings viel seltener als die mit Funktionen.

Die Anwendung schließen

Als Erstes sollten Sie sich darum kümmern, dass der Benutzer die Anwendung schließen kann. Da Sie die Schaltflächen OK und ABBRECHEN gelöscht und eine neue Schaltfläche für das Schließen des Anwendungsfensters hinzugefügt haben, müssen Sie Code in die Funktion aufnehmen, die von der Schaltfläche EXIT ausgelöst wird, um das Fenster zu schließen. Führen Sie dazu die folgenden Schritte aus:

1. Verwenden Sie die Registerkarte über dem Bearbeitungsbereich, um das Dialogfenster zu öffnen, und wählen Sie die Schaltfläche EXIT aus. Fügen Sie mithilfe der Eigenschaftenansicht eine Funktion auf der `BN_CLICKED`-Nachricht für `button` ein, wie Sie es an den vergangenen beiden Tagen gelernt haben. Wenn Sie den Dialog geschlossen oder das Projekt geschlossen und neu geöffnet haben, verwenden Sie die Ressourcen-Ansicht, um den Dialog wieder zu öffnen.
2. Geben Sie den fett gedruckten Code aus Listing 3.2 ein.

Listing 3.2: Die Funktion OnBnClickedExit

```
1: void CControlsDlg::OnBnClickedExit()
2: {
3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
4:   // Benachrichtigung ein.
```

```
5:   // Das Programm verlassen
6:   OnOK();
7: }
```

Ein einziger Funktionsaufruf in der Funktion `OnBnClickedExit` schließt das Fenster und beendet die Anwendung. Woher kommt diese `OnOK`-Funktion und warum mussten Sie sie nicht in der gestrigen Anwendung aufrufen? Zwei Funktionen, `OnOK` und `OnCancel`, sind in der Basisklasse `CDialog` definiert, von der Ihre Klasse `CControlsDlg` abgeleitet ist. In der Klasse `CDialog` hat die Nachrichtenzuordnungstabelle bereits die Objekt-IDs der Schaltflächen OK und ABBRECHEN mit den Funktionen `OnOK` bzw. `OnCancel` verbunden, sodass Schaltflächen mit diesen IDs automatisch die entsprechenden Funktionen aufrufen. Wenn Sie die Objekt-ID der Schaltfläche EXIT mit `IDOK` festgelegt hätten, müssten Sie der Schaltfläche keinerlei Code zuordnen, solange Sie nicht die grundlegende Funktionalität von `OnOK` überschrieben.

Sie fragen sich vielleicht, warum Sie in diesem Fall nicht durch diesen Aufruf angeben mussten, dass sich die Funktion `OnOK` in der Basisklasse `CDialog` befindet:

```
CDialog::OnOK();
```

Sie mussten das nicht angeben, weil die Funktion in Ihrer Klasse nicht überschrieben wird. Wenn Sie eine Funktion aus einer Basisklasse aufrufen, die nicht in einer abgeleiteten Klasse überschrieben wurde, müssen Sie den Scope-Resolution-Operator nicht verwenden, um die Klasse anzugeben, in der die Funktion enthalten ist.

Die Nachricht des Benutzers anzeigen

Es sollte ein Leichtes sein, die vom Benutzer in das Eingabefeld eingegebene Nachricht anzuzeigen, da dieser Vorgang den gestern beschriebenen Abläufen ähnelt. Sie können der Schaltfläche ZEIGE MITTEILUNG eine Funktion zuordnen und die Funktion `MessageBox` aufrufen, wie es aus Listing 3.3 hervorgeht.

Listing 3.3: Die Funktion OnBnClickedShwmsg zeigt die Benutzernachricht an.

```
void CControlsDlg::OnBnClickedShwmsg()
{
  // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
  // Benachrichtigung ein.
  // Die Nachricht des Benutzers anzeigen
  MessageBox(m_strMessage);
}
```

Wenn Sie die Anwendung in dieser Phase kompilieren und ausführen, tritt ein Problem mit diesem Code zutage. Es erscheint der String, mit dem Sie die Variable `m_strMessage` in der Funktion `OnInitDialog` initialisiert haben, und nicht die Nachricht, die der Benutzer in das Eingabefeld eingibt. Das ist darauf zurückzuführen, dass Sie die Variable noch nicht mit dem Inhalt des Steuerelements im Fenster aktualisiert haben. Es ist `UpdateData` mit dem Argument `TRUE` aufzurufen, um die Werte der Steuerelemente zu lesen und die Variablen damit zu initialisieren, bevor Sie die Funktion `MessageBox` aufrufen. Ändern Sie die Funktion `OnBnClickedShwmsg` gemäß Listing 3.4 ab.

Listing 3.4: Die aktualisierte Version der Funktion OnBnClickedShwMsg

```
 1: void CControlsDlg::OnBnClickedShwmsg()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Nachrichtenvariable mit der Benutzereingabe aktualisieren
 6:   UpdateData(TRUE);
 7:
 8:   // Die Nachricht des Benutzers anzeigen
 9:   MessageBox(m_strMessage);
10: }
```

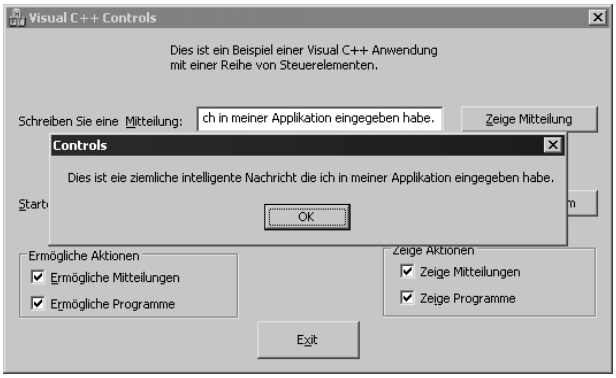

Abbildung 3.8: Die in das Eingabefeld eingetippte Nachricht wird dem Benutzer angezeigt.

Wenn Sie Ihre Anwendung jetzt kompilieren und ausführen, sollte die in das Eingabefeld eingegebene Meldung angezeigt werden, wie es Abbildung 3.8 verdeutlicht.

Die Nachricht des Benutzers löschen

Falls der Benutzer ein leeres Eingabefeld vorfinden möchte, bevor er eine Nachricht eintippt, können Sie der Schaltfläche LÖSCHE MITTEILUNG eine Funktion zuordnen, um den Inhalt des Eingabefeldes zu leeren. Die Funktion fügen Sie in der gewohnten Weise über den Class Assistent hinzu. Die Funktionalität realisieren Sie ganz einfach, indem Sie die Variable `m_strMessage` auf einen leeren String setzen und dann die Steuerelemente im Fenster aktualisieren, um diesen Wert wiederzugeben. Der entsprechende Code ist in Listing 3.5 zu sehen.

Listing 3.5: Die Funktion OnBnClickedClrmsg

```
 1: void CControlsDlg::OnBnClickedClrmsg()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Nachricht löschen
 6:   m_strMessage = "";
 7:
 8:  // Bildschirm aktualisieren
 9:   UpdateData(FALSE);
10: }
```

Die Nachrichtensteuerelemente deaktivieren und ausblenden

In Bezug auf die Nachrichtensteuerelemente ist als Letztes noch die Funktionalität für die Kontrollkästchen ERMÖGLICHE MITTEILUNGEN und ZEIGE MITTEILUNGEN zu implementieren. Das erste dieser Kontrollkästchen aktiviert oder deaktiviert die Steuerelemente, die sich auf die Anzeige der Benutzernachricht beziehen. Wenn das Kontrollkästchen eingeschaltet ist, sind alle Steuerelemente aktiviert. Weist das Kontrollkästchen den ausgeschalteten Zustand auf, sind die betreffenden Steuerelemente deaktiviert. Analog dient das zweite Kontrollkästchen dazu, dieselbe Gruppe der Steuerelemente anzuzeigen bzw. auszublenden. Listing 3.6 zeigt den Code für beide Funktionen.

Listing 3.6: Die Funktionen für die Kontrollkästchen Ermögliche Mitteilungen und Zeige Mittteilungen

```
 1: void CControlsDlg::OnBnClickedCkenblmsg()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Aktuelle Werte vom Bildschirm holen
```

```
 6:   UpdateData(TRUE);
 7:
 8:   // Kontrollkästchen 'Ermögliche Mitteilungen' eingeschaltet?
 9:   if (m_bEnableMsg == TRUE)
10:   {
11:     // Ja, dann alle Steuerelemente aktivieren,
12:     // die für die Anzeige der Nachricht relevant sind.
13:     GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
14:     GetDlgItem(IDC_SHWMSG)->EnableWindow(TRUE);
15:     GetDlgItem(IDC_DFLTMSG)->EnableWindow(TRUE);
16:     GetDlgItem(IDC_CLRMSG)->EnableWindow(TRUE);
17:     GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
18:   }
19:   else
20:   {
21:     // Nein, dann alle Steuerelement deaktivieren,
22:     // die für die Anzeige der Nachricht relevant sind.
23:     GetDlgItem(IDC_MSG)->EnableWindow(FALSE);
24:     GetDlgItem(IDC_SHWMSG)->EnableWindow(FALSE);
25:     GetDlgItem(IDC_DFLTMSG)->EnableWindow(FALSE);
26:     GetDlgItem(IDC_CLRMSG)->EnableWindow(FALSE);
27:     GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
28:   }
29: }
30:
31: void CControlsDlg::OnBnClickedCkshwmsg()
32: {
33:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
34:   // Benachrichtigung ein.
35:   // Aktuelle Werte vom Bildschirm holen
36:   UpdateData(TRUE);
37:   // Kontrollkästchen 'Zeige Mittteilungen' aktiviert?
38:   if (m_bShowMsg == TRUE)
39:   {
40:     // Ja, dann alle Steuerelemente anzeigen, die
41:     // für Anzeige der Nachricht relevant sind.
42:     GetDlgItem(IDC_MSG)->ShowWindow(TRUE);
43:     GetDlgItem(IDC_SHWMSG)->ShowWindow(TRUE);
44:     GetDlgItem(IDC_DFLTMSG)->ShowWindow(TRUE);
45:     GetDlgItem(IDC_CLRMSG)->ShowWindow(TRUE);
46:     GetDlgItem(IDC_STATICMSG)->ShowWindow(TRUE);
47:   }
48:   else
49:   {
50:     // Nein, dann alle Steuerelemente ausblenden, die
51:     // für Anzeige der Nachricht relevant sind.
```

```
52:     GetDlgItem(IDC_MSG)->ShowWindow(FALSE);
53:     GetDlgItem(IDC_SHWMSG)->ShowWindow(FALSE);
54:     GetDlgItem(IDC_DFLTMSG)->ShowWindow(FALSE);
55:     GetDlgItem(IDC_CLRMSG)->ShowWindow(FALSE);
56:     GetDlgItem(IDC_STATICMSG)->ShowWindow(FALSE);
57:   }
58: }
```

Den ersten Teil dieser Funktionen sollten Sie mittlerweile verstehen. Als Erstes werden die Variablen mit den aktuellen Werten der Steuerelemente im Fenster aktualisiert. Es folgt ein Test der Booleschen Variablen, die mit dem jeweiligen Kontrollkästchen verbunden ist. Enthält die Variable den Wert `TRUE`, soll das Programm das Steuerelement aktivieren oder anzeigen. Hat die Variable den Wert `FALSE`, ist das Steuerelement zu deaktivieren bzw. auszublenden.

Von jetzt an ist der Code nicht mehr ganz so durchsichtig. Die erste Funktion, `GetDlgItem`, erhält als Parameter die ID des zu ändernden Steuerelements. Die Funktion liefert einen Zeiger auf das Steuerelement zurück. Dadurch lässt sich ein Zeiger auf beliebige Steuerelemente des Fensters bei laufender Anwendung abrufen. Der nächste Teil jedes Befehls ruft eine Member-Funktion des Steuerelementobjekts auf. Die zweite Funktion ist eine Member-Funktion des Steuerelements, für das von der ersten Funktion ein Zeiger zurückgegeben wurde.

Die zweiten Funktionen in diesen Aufrufen, `EnableWindow` und `ShowWindow`, sehen eher wie Funktionen für Fenster, aber nicht für Steuerelemente, aus. Natürlich sind sie für Fenster vorgesehen. Zufällig sind Steuerelemente aber auch Elemente der Klasse `CWnd`, die eine Basisklasse der Klasse `CDialog` ist, von der Sie Ihre Klasse `CControlsDlg` abgeleitet haben. In Windows sind nun mal alle Steuerelemente selbst Fenster, die völlig unabhängig von dem Fenster sind, in dem sie sich befinden. Damit kann man Steuerelemente als Fenster behandeln und ihre Fensterfunktionen aufrufen. In der Tat sind alle Steuerelementklassen von der Klasse `CWnd` abgeleitet, was ihr wahres Gesicht als Fenster offenbart.

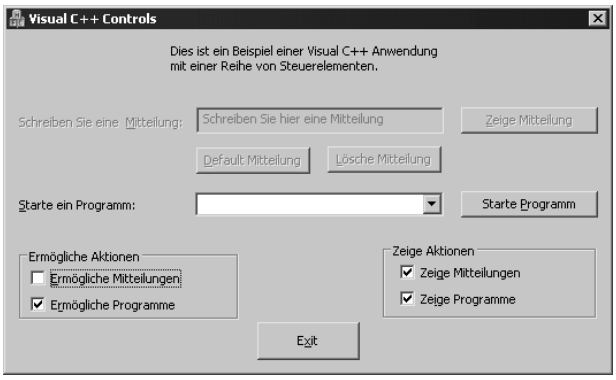

Abbildung 3.9:
Die Steuerelemente für Benutzermitteilungen lassen sich nun deaktivieren.

Wenn Sie jetzt Ihre Anwendung kompilieren und ausführen, können Sie die Kontrollkästchen ERMÖGLICHE MITTEILUNGEN und ZEIGE MITTEILUNGEN ausprobieren. Die Funktionsweise sollte Abbildung 3.9 entsprechen.

C++-Exkurs: Boolesche Ausdrücke

In Listing 3.6 hätten Sie die `if`-Statements auch etwas anders schreiben können. Da die Booleschen Variablen, die Sie überprüfen, von ihrer Natur her entweder `TRUE` oder `FALSE` sind, müssen Sie ihren Wert nicht wirklich mit `TRUE` vergleichen. Stattdessen können Sie das `if`-Statement einfach so schreiben:

```
if (m_bEnableMsg)
```

Nach dieser Logik können Sie das Gleiche mit verschiedenen Variablen tun, bei denen Sie nur feststellen müssen, ob diese Variablen einen Wert besitzen (also beispielsweise nicht `0` sind). Das wird oft bei der Überprüfung von Zeigern verwendet, um sicherzustellen, dass sie auf etwas zeigen. Dabei wird immer angenommen, dass die fragliche Variable mit `NULL` (als 0 definiert) initialisiert wurde. Wenn Sie beispielsweise überprüfen möchten, ob einer Variable ein Wert zugewiesen wurde, könnten Sie das so tun:

```
if (iVar != 0)
```

Sie könnten jedoch auch Folgendes verwenden:

```
if (iVar)
```

In Umkehrung dieser Logik können Sie auch überprüfen, ob eine Variable gleich `0` ist, indem Sie wie folgt den Ausdruck negieren:

```
if (!iVar)
```

In diesem Fall bedeutet das Ausrufezeichen (`!`) *nicht*. Genau wie `!=` »nicht gleich« bedeutet, liest sich `!iVar` als »nicht `iVar`« oder »`iVar` ist gleich null«.

C++-Exkurs: Objektzeiger

Der nächste Teil des obigen Code-Listings besteht in erster Linie aus einer Wiederholung ein und derselben Zeile. Diese Zeile sieht so aus:

```
GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
```

Der ersten Funktion, `GetDlgItem`, wird die ID eines Steuerelements im Dialog übergeben. Die Funktion gibt einen Zeiger auf das angegebene Steuerelement zurück. Wenn eine Funktion einen Zeiger zurückgibt, können Sie diesen in einer Variable speichern oder wie in diesem Beispiel den zurückgegebenen Zeiger verwenden, um eine Member-Funktion des Objekts aufzurufen, auf das der Zeiger verweist.

Ein *Zeiger* (auch *Pointer* genannt) ist ein Verweis auf ein Objekt oder eine Variable. Er besteht aus der Speicheradresse, an der sich das Objekt oder die Variable derzeit befindet. Wenn Sie einen Zeiger verwenden, anstatt der CPU das Objekt zu übergeben, teilen Sie der CPU mit, wo sie das Objekt findet. Dadurch sind Zeiger wesentlich kleiner und somit schneller verarbeitbar als umfangreiche Datenstrukturen.

Wenn Sie eine Variable von einer bestimmten Klasse deklarieren, können Sie mithilfe der Punkt-Notation die Methoden dieser Klasse aufrufen und auf ihre Eigenschaften und Variablen zugreifen:

```
CMyObject obj;
obj.Method1();
```

Wenn Sie mit einem Zeiger arbeiten, ändert sich die Syntax an zwei Stellen. Zuerst wird ein Zeiger mit einem Stern (`*`) vor dem Variablennamen deklariert:

```
CMyObject *obj;
```

oder

```
CMyObject* obj;
```

Die erste Deklaration sagt aus, dass die Variable `obj` ein Zeiger auf eine Klasse `CMyObject` ist. Diese Zeile könnte auch noch eine weitere Variable enthalten, die eine Instanz der Klasse `CMyObject` ist:

```
CMyObject *obj, objInstance;
```

Die zweite Deklaration sagt aus, dass es sich bei allen in dieser Zeile deklarierten Variablen um Zeiger auf Instanzen der Klasse `CMyObject` handelt.

Wenn Sie einen Zeiger verwenden, um auf Member-Methoden, Eigenschaften oder Variablen eines Objekts zuzugreifen, verwenden Sie immer die `->`-Notation:

```
obj->Method1();
```

Wenn Sie eine Instanz eines Objekts oder einer Variablen haben und einen Zeiger initialisieren wollen, der auf diese Instanz verweist, verwenden Sie das kaufmännische Und (`&`), um die Adresse des Objekts zu erhalten:

```
CMyObject objInstance, *pObj;
pObj = &objInstance;
```

Ein kaufmännisches Und vor einer Variablen wird als »Adresse von« gelesen, also setzen Sie im obigen Code den Wert von `pObj`, einem Zeiger, auf die Adresse von `objInstance`, einer Instanz der Klasse `CMyObject`.

Wenn wir zu unserem Beispielcode zurückkommen,. Kann die sich wiederholende Code-Zeile auch so geschrieben werden:

```
Cwnd *pWnd;

pWnd = getDlgItem(IDC_MSG);
pWnd->EnableWindow(TRUE);
```

Sie können diese Funktionalität jedoch auf eine einzelne Zeile verkürzen, indem Sie die Möglichkeit nutzen, Funktionalität in einer Code-Zeile zu vereinigen und wie in Listing 3.6 den von der Funktion `GetDlgItem` zurückgegebenen Zeiger verwenden, um auf die Member-Funktion `EnableWindow` des angegebenen Steuerelements zuzugreifen:

```
GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
```

Wenn man diese Fähigkeit von C++, Funktionalität zu kombinieren, noch weiter treibt, kann die Größe der beiden Funktionen in Listing 3.6 verkürzt werden, indem man das `if`-Statement herausnimmt und die Boolesche Variable an die Funktionen `EnableWindow` und `ShowWindow` übergibt, anstatt die Parameterwerte hart einzuprogrammieren. So könnten Sie diesen Code nehmen:

```
// Kontrollkästchen 'Enable Message Action' eingeschaltet?
if (m_bEnableMsg == TRUE)
{
  // Ja, dann alle Steuerelemente aktivieren,
  // die für Anzeige der Nachricht relevant sind.
  GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
  GetDlgItem(IDC_SHWMSG)->EnableWindow(TRUE);
  GetDlgItem(IDC_DFLTMSG)->EnableWindow(TRUE);
  GetDlgItem(IDC_CLRMSG)->EnableWindow(TRUE);
  GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
}
else
{
  // Nein, dann alle Steuerelemente deaktivieren,
  // die für Anzeige der Nachricht relevant sind.
  GetDlgItem(IDC_MSG)->EnableWindow(FALSE);
  GetDlgItem(IDC_SHWMSG)->EnableWindow(FALSE);
  GetDlgItem(IDC_DFLTMSG)->EnableWindow(FALSE);
  GetDlgItem(IDC_CLRMSG)->EnableWindow(FALSE);
  GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
}
```

und auf Folgendes reduzieren:

```
// Alle für die Anzeige der Benutzernachricht relevanten
// Steuerelemente aktivieren oder deaktivieren
GetDlgItem(IDC_MSG)->EnableWindow(m_bEnableMsg);
```

```
GetDlgItem(IDC_SHWMSG)->EnableWindow(m_bEnableMsg);
GetDlgItem(IDC_DFLTMSG)->EnableWindow(m_bEnableMsg);
GetDlgItem(IDC_CLRMSG)->EnableWindow(m_bEnableMsg);
GetDlgItem(IDC_STATICMSG)->EnableWindow(m_bEnableMsg);
```

MFC-Exkurs: Member-Methoden von `CWnd`

In der Klasse `CWnd`, der Basisklasse für alle Benutzerschnittstellenobjekte in MFC, wurden in Listing 3.6 drei Member-Methoden verwendet:

- `GetDlgItem`
- `EnableWindow`
- `ShowWindow`

Die erste dieser Methoden, `GetDlgItem`, kann verwendet werden, um einen Zeiger auf ein Kindfenster abzurufen. Der zurückgegebene Zeiger ist ein `CWnd`-Zeiger. Dieser Methode wird ein Parameter übergeben: die `ID` des Kindfensters. Gewöhnlich wird diese Funktion verwendet, um einen Zeiger auf ein Steuerelement oder einen Dialog abzurufen, doch man kann mit ihr einen Zeiger auf jedes beliebige Kindfenster bekommen.

Die zweite Methode, `EnableWindow`, schaltet die Möglichkeit, dass der Benutzer mit dem Fenster, für das sie aufgerufen wurde, interagieren kann, ein oder aus. Sie übernimmt einen Booleschen Wert als einzigen Parameter. Ist der Parameter `TRUE`, so ist das Fenster aktiviert und der Benutzer kann damit interagieren. Ist der Parameter `FALSE`, so ist das Fenster oder Steuerelement deaktiviert und ignoriert Benutzeraktionen.

Die dritte Methode, `ShowWindow`, ähnelt der Methode `EnableWindow`. Sie verbirgt oder zeigt das Fenster oder Steuerelement. Sie übernimmt einen Booleschen Wert als Parameter, der ihr mitteilt, ob das Fenster oder Steuerelement sichtbar sein soll oder nicht.

Eine andere Anwendung starten

Als letzte größere Aufgabe müssen wir noch die Funktionalität für die Steuerelemente implementieren, die ein anderes Programm starten. Oberhalb von Abbildung 3.3 haben Sie die Namen von drei Windows-Anwendungen in das Kombinationsfeld aufgenommen. Wenn Sie die Anwendung starten, erscheinen diese Namen in der DropDown-Liste. Man kann einen Eintrag auswählen und im Wertbereich des Kombinationsfelds erscheint der Name der jeweiligen Anwendung. Damit die auch eine echte Funktion ausführt, müssen Sie lediglich für die Schaltfläche STARTE PROGRAMM Code hinzufügen, um tatsächlich den Wert aus dem Kombinationsfeld zu ermitteln und das passende Programm zu starten.

Nachdem Sie das Funktionsgerüst für die Schaltfläche STARTE PROGRAMM erstellt haben, fügen Sie den Code aus Listing 3.7 in die Funktion ein.

Listing 3.7: Die Funktion OnBnClickedRunpgm startet andere Windows-Anwendungen.

```
1: void CControlsDlg::OnBnClickedRunpgm()
2: {
3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
4:   //  Benachrichtigung ein.
5:   // Aktuelle Werte vom Bildschirm holen
6:   UpdateData(TRUE);
7:
8:   // Lokale Variable zur Aufnahme des Programmnamens deklarieren
9:   CString strPgmName;
10:
11:  // Programmname in die lokale Variable kopieren
12:  strPgmName = m_strProgToRun;
13:
14:  // Programmname in Großbuchstaben umwandeln
15:  strPgmName.MakeUpper();
16:
17:  // Programm Paint gewählt?
18:  if (strPgmName == "PAINT")
19:    // Ja, Paint starten
20:    WinExec("mspaint.exe ", SW_SHOW);
21:
22:  // Programm Notepad (Editor) gewählt?
23:  if (strPgmName == "NOTEPAD")
24:    // Ja, Notepad starten
25:    WinExec("notepad.exe ", SW_SHOW);
26:
27:  // Programm Solitaire gewählt?
28:  if (strPgmName == "SOLITAIRE")
29:    // Ja, Solitaire starten
30:    WinExec("sol.exe ", SW_SHOW);
31: }
```

Wie zu erwarten, findet in dieser Funktion zunächst der Aufruf von `UpdateData` statt, um die Variablen mit den Werten der Steuerelemente im Fenster zu aktualisieren. Der nächste Teil erscheint allerdings ein wenig eigentümlich. Es wird eine neue `CString`-Variable deklariert und in diese der Wert des Kombinationsfeldes kopiert. Ist es wirklich notwendig, wenn der Wert bereits in einer `CString`-Variablen steht?

Das hängt davon ab, wie sich Ihre Anwendung verhalten soll. Die nächste Code-Zeile enthält einen Aufruf der `CString`-Funktion `MakeUpper`, die den String in Großbuchstaben konvertiert. Wenn man die `CString`-Variable verwendet, die mit dem Kombinationsfeld

verbunden ist, wird beim nächsten Aufruf von `UpdateData` mit dem Argument `FALSE` der Wert im Kombinationsfeld ebenfalls in Großbuchstaben angezeigt. Unter diesen Umständen ist das wahrscheinlich ein ungünstiger Zeitpunkt und das Verhalten entspricht nicht den Vorstellungen. Deshalb kommt in der Funktion ein zusätzlicher `CString` zum Einsatz.

An die Umwandlung des Strings in Großbuchstaben schließt sich eine Folge von `if`-Anweisungen an, die den String mit den Namen der verschiedenen Programme vergleichen. Bei einer gefundenen Übereinstimmung ruft der Code die Funktion `WinExec` auf, um die betreffende Anwendung zu starten. Wenn Sie Ihre Anwendung jetzt kompilieren und ausführen, können Sie eine der Anwendungen aus der DropDown-Liste auswählen und sie durch Klicken auf die Schaltfläche STARTE PROGRAMM öffnen.

In C und C++ ist der Unterschied zwischen einem einfachen Gleichheitszeichen (=) und einem doppelten Gleichheitszeichen (==) zu beachten. Das einfache Gleichheitszeichen führt eine Zuweisung des Wertes auf der rechten Seite des Gleichheitszeichens an die Variable auf der linken Seite durch. Steht auf der linken Seite des Gleichheitszeichens eine Konstante, wird das Programm nicht kompiliert und Sie erhalten die Fehlermeldung, dass man keinen Wert der rechten Seite an eine Konstante auf der linken Seite zuweisen kann. Das doppelte Gleichheitszeichen ist für Vergleiche vorgesehen. Achten Sie darauf, das doppelte Gleichheitszeichen zu verwenden, wenn Sie zwei Werte miteinander vergleichen möchten. Wenn Sie in diesem Fall versehentlich das einfache Gleichheitszeichen verwenden, ändern Sie den Wert der Variablen auf der linken Seite. Hier liegt eine der größten Quellen für logische Fehler in C/C++-Programmen.

`WinExec` ist eine veraltete Windows-Funktion. Man sollte statt dessen die Funktion `CreateProcess` verwenden. Allerdings weist `CreateProcess` eine Reihe von Argumenten auf, die in dieser frühen Phase der Programmierung mit Visual C++ nicht so leicht zu verstehen sind. Die Funktion `WinExec` ist weiterhin verfügbar und als Makro implementiert, das die Funktion `CreateProcess` aufruft. Damit können Sie auf die wesentlich einfachere Funktion `WinExec` zurückgreifen, um eine andere Anwendung zu starten, wobei gesichert ist, dass Windows die eigentlich erwartete Funktion benutzt.

Die API-Funktion `ShellExecute` lässt sich ebenfalls für den Start einer anderen Anwendung einsetzen. Diese Funktion war ursprünglich vorgesehen, um Dateien zu öffnen oder zu drucken, man kann sie aber auch zum Starten anderer Programme nutzen.

3.5 Zusammenfassung

Heute haben Sie gelernt, wie man Standardsteuerelemente von Windows in einer Visual C++-Anwendung einsetzt. Es wurde gezeigt, wie man Variablen deklariert, sie mit den Steuerelementen verbindet und wie man die Werte der Variablen mit den Steuerelementen synchronisiert. Weiterhin haben Sie gelernt, wie man Steuerelemente manipuliert, indem man die Steuerelementobjekte mittels ihrer Objekt-ID abruft, und wie man das Steuerelement manipuliert, indem man es als Fenster behandelt. Als Nächstes wurde auf die Tabulator-Reihenfolge der Steuerelemente in einer Anwendung eingegangen. Damit legt man die Richtung und Reihenfolge fest, in welcher der Benutzer durch eine Windows-Anwendung navigiert. Schließlich haben Sie gelernt, wie man die Funktionalität einer Anwendung mit den Steuerelementen im Anwendungsfenster verbindet, wobei die verschiedenen Aktionen ausgelöst werden, wenn der Benutzer mit den jeweiligen Steuerelementen interagiert. Als Bonus wurde gezeigt, wie man andere Windows-Anwendungen aus der eigenen Anwendung heraus aufrufen kann.

3.6 Workshop

Fragen und Antworten

F *Nachdem ich die Objekt-IDs der Steuerelemente im Fenster festgelegt habe, weisen drei Steuerelemente dieselbe ID,* `IDC_STATIC`*, auf. Bei diesen Steuerelementen handelt es sich um den Text im oberen Teil des Fensters und um die beiden Gruppenfelder. Die zwei anderen statischen Textsteuerelemente hatten zuerst die gleiche ID, bis ich sie geändert habe. Wieso haben diese Steuerelemente die gleiche ID und warum musste ich die IDs der beiden statischen Textsteuerelemente an dieser Stelle ändern?*

A Alle Steuerelemente, die normalerweise nicht für eine Benutzerinteraktion vorgesehen sind, beispielsweise statischer Text und Gruppenfelder, erhalten per Vorgabe die gleiche Objekt-ID. Das funktioniert, solange Ihre Anwendung keine Aktionen auf diesen Steuerelementen ausführen muss. Wenn Sie eine Interaktion mit einem dieser Steuerelemente beabsichtigen, wie es für den statischen Text mit den Aufforderungen für das Eingabefeld und das Kombinationsfeld geschehen ist, müssen Sie den betreffenden Steuerelementen eine eindeutige ID zuweisen. In diesem Fall war die eindeutige ID erforderlich, damit Sie das Steuerelementobjekt abrufen können, um das Steuerelement zu aktivieren/zu deaktivieren bzw. anzuzeigen/auszublenden. Ebenfalls müssen Sie eine eindeutige ID zuweisen, wenn Sie eine Variable mit einem Steuerelement verbinden wollen, um die Beschriftung auf dem Steuerelement dynamisch ändern zu können.

A Die Anwendung verhält sich in unvorhergesehener Weise, wenn Sie eines der statischen Steuerelemente, welche die gleiche ID haben, verändern. Als Faustregel gilt, dass man statischen Steuerelementen die gleiche ID zuweisen kann, wenn man die Steuerelemente überhaupt nicht ändert. Wird eine Interaktion mit den Steuerelementen erforderlich, müssen Sie jedem Steuerelement eine eindeutige Objekt-ID zuordnen.

F *Gibt es eine andere Möglichkeit, die Steuerelemente zu manipulieren, als die Steuerelementobjekte unter Verwendung ihrer Objekt-IDs abzurufen?*

A Im Assistent zum Hinzufügen von Membervariablen können Sie Variablen für Ihre Steuerelemente deklarieren, indem Sie als Variablenkategorie CONTROL festlegen. Damit erhält man grundsätzlich ein Objekt, das die MFC-Klasse des Steuerelements darstellt, und man kann dann direkt das Steuerelement ändern und damit interagieren. Für das Steuerelement lassen sich dann alle Funktionen der Klasse `CWnd` aufrufen, wie Sie es beim Aktivieren/Deaktivieren bzw. Anzeigen/Ausblenden der Steuerelemente in Ihrer Anwendung vorgenommen haben. Sie können auch die Klassenmethoden der Steuerelemente aufrufen. Das bietet Ihnen die Möglichkeit, im Code spezielle Aufgaben für den jeweiligen Steuerelementtyp zu realisieren. Wenn Sie zum Beispiel dem Kombinationsfeld eine weitere Variable zuordnen und festlegen, dass es sich um eine Variable der Kategorie CONTROL handelt, können Sie über diese Variable die Elemente in der Drop-Down-Liste des Steuerelements hinzufügen.

Quiz

1. Warum muss man die Tabulator-Reihenfolge der Steuerelemente im Anwendungsfenster festlegen?
2. Wie kann man eine Zugriffstaste einbinden, die den Benutzer zum Eingabefeld oder Kombinationsfeld bringt?
3. Warum muss man den statischen Textfeldern vor dem Eingabefeld und vor den Kombinationsfeldern eindeutige Objekt-IDs zuweisen?
4. Warum muss man die Funktion `UpdateData` aufrufen, bevor man den Wert eines Steuerelements überprüft?

Übungen

1. Fügen Sie für die Schaltfläche DEFAULT MITTEILUNG Code hinzu, um den Text im Eingabefeld auf »Schreiben Sie hier eine Mitteilung« zurückzusetzen.
2. Nehmen Sie Code auf, um die Steuerelemente zur Auswahl und zum Start einer anderen Anwendung zu aktivieren oder zu deaktivieren sowie anzuzeigen oder auszublenden.
3. Erweitern Sie den Code in der Funktion `OnBnClickedRunpgm` (in Listing 3.7), damit der Benutzer den Namen des auszuführenden Programms selbst eingeben kann.