Kapitel 5

Modale Fenster, nicht-modale Fenster, Ereignisse

In diesem Kapitel erfahren Sie

- wie Sie im Code auf Benutzeraktionen reagieren
- was Ereignisprogrammierung bedeutet
- wie Sie in Ihren Programmen modale und nicht-modale Dialogfelder einrichten
- wie Sie Meldungsfenster implementieren

5.1 Ereignisprogrammierung

Programme geben nicht nur Text aus, sondern reagieren im Allgemeinen auch auf Benutzereingaben und andere Aktionen. Mit der Datenverarbeitung werden wir uns in Kapitel 6 ausführlich beschäftigen. Hier soll es vor allem um so genannte Ereignisse gehen. Die ganze Windows-Programmierung im .NET Framework ist untrennbar verknüpft mit der Ereignisprogrammierung.

Vielleicht sollten wir zunächst einmal klären, was ein Ereignis überhaupt ist. Unter Windows und im .NET Framework gibt es eine ganze Reihe von Ereignissen. Sie treten immer zur Laufzeit eines Programms auf. In der Regel werden sie durch ein ganz bestimmtes Benutzerverhalten ausgelöst wie etwa das Setzen des Cursors in ein Textfeld, das Überfahren eines Beschriftungsfeldes mit der Maus, das Anklicken einer Schaltfläche usw.

Bleiben wir bei der Schaltfläche. Wenn der Anwender darauf klickt, tritt das so genannte `Click`-Ereignis ein. Wie Sie sehen, haben Ereignisse auch einen Namen. Den können Sie zum Beispiel auch im Eigenschaftenfenster sehen. (Wir zeigen Ihnen gleich im nächsten Abschnitt, wo dort die Namen von Ereignissen zu finden sind.)

Was uns Visual C++-Programmierer aber vor allem interessiert, sind die Methoden, die zur Ausführung kommen, sobald ein Ereignis eintritt. Auch das `Click`-Ereignis ist mit einer Methode verknüpft. Immer dann, wenn ein Benutzer eine Schaltfläche anklickt, löst er damit das jeweilige Ereignis aus und die zugehörige Methode wird ausgeführt.

Das impliziert, dass die gleichen Ereignisse während eines Programmlaufs wiederholt auftreten können. Wenn der Benutzer drei Mal auf dieselbe Schaltfläche klickt, löst er das `Click`-Ereignis damit auch drei Mal aus. Wenn er das zwanzig Mal macht, eben zwanzig Mal, und jedes Mal wird die zugehörige Methode ausgeführt.

Solche Methoden, die automatisch ausgeführt werden, wenn ein bestimmtes Ereignis eintritt, genießen übrigens mehrere Bezeichnungen. Man nennt sie Ereignismethoden, Eventhandler oder Ereignishandler. Sie können auch von Ereignisbehandlungsmethoden bzw. Ereignisbehandlungsroutinen sprechen, was deren Funktion vielleicht deutlicher macht. Schließlich bilden diese Methoden für Sie als Programmierer eine einfache und übersichtliche Möglichkeit, um im Code auf Benutzeraktionen zu reagieren.

Nun ist es keine Seltenheit, dass die Benutzeroberfläche einer Anwendung mehrere Schaltflächen – und natürlich andere Steuerelemente – enthält. Löst ein Klick auf die Schaltfläche mit der Aufschrift *Weiter* in diesem Fall dasselbe Ereignis aus wie ein Klick auf die Schaltfläche *Abbrechen*?

Die Antwort ist nein. Denken Sie daran, dass ein Eventhandler – die alternative Bezeichnung Ereignisbehandlungsroutine macht es hier deutlicher – für Sie als Programmierer da ist. Sie erlaubt es Ihnen, im Code auf ein spezielles Ereignis zu reagieren. Zur Verdeutlichung noch einmal anders ausgedrückt: auf ein bestimmtes Benutzerverhalten zu reagieren, denn dieses löst ja das Ereignis aus. Und selbstverständlich möchten Sie für den Fall, dass ein Benutzer auf die *Weiter*-Schaltfläche klickt, andere Programmierbefehle, sprich: Anweisungen, vorsehen als für den Fall, dass die Schaltfläche *Abbrechen* angeklickt wird.

Also: Jede Schaltfläche einer grafischen Benutzeroberfläche (GUI) hat im .NET Framework ihr eigenes `Click`-Ereignis, das mit einer speziellen Methode verbunden ist. Letztere wird automatisch immer dann ausgeführt, wenn ein Benutzer das betreffende Ereignis durch einen Klick auf die Schaltfläche ausgelöst hat.

Wenn eine Methode ausgeführt wird, heißt das, dass die Anweisungen, die sich im Rumpf der Methode – zwischen den beiden geschweiften Klammern (`{ ... }`) – befinden, ausgeführt werden. Nach dem Gesagten ergibt sich: Sie als Programmierer können eine Ereignismethode praktisch nach Lust und Laune mit Programmcode, sprich: Anweisungen, füllen und damit bestimmen, was geschieht, wenn ein Benutzer durch sein Verhalten das betreffende Ereignis auslöst.

Gehen wir daran, das genannte Beispiel mit der Schaltfläche und dem `Click`-Ereignis – und der zugehörigen Ereignismethode – gedanklich auszuweiten. Bleiben wir zunächst noch bei der Schaltfläche. Diese besitzt nämlich nicht nur das `Click`-Ereignis, sondern noch weitere. Zum Beispiel `Resize` oder `Enter`, um nur zwei davon zu nennen – tatsächlich sind es noch viel mehr.

Das Ereignis `Resize` tritt ein, wenn sich die Größe einer Schaltfläche ändert – etwa dadurch, dass der Benutzer das übergeordnete Fenster minimiert oder maximiert. Das Schaltflächenereignis `Enter` wird ausgelöst, wenn eine Schaltfläche den Fokus erhält.

Wie Sie sich vermutlich denken können, kann ein bestimmtes Benutzerverhalten gleich zwei oder gar noch mehr Ereignisse auslösen. Wenn ein Benutzer eine Schaltfläche anklickt, dann erhält diese natürlich den Fokus, falls sie ihn zu diesem Zeitpunkt noch nicht hatte. In diesem Fall löst der Mausklick sowohl das Ereignis `Click` als auch das Ereignis `Enter` aus. Zur Ausführung kommen dann – in einer bestimmten Reihenfolge – beide Ereignismethoden.

Die Reihenfolge ist tatsächlich fix, da der Zeitpunkt bzw. die Voraussetzungen, wann ein Ereignis eintritt, genau definiert sind. Auch wenn eine Benutzeraktion mehrere Ereignisse auslösen kann, treten diese Ereignisse dennoch zeitlich versetzt auf, nie simultan.

Für obigen Fall müssen Sie als Programmierer gegebenenfalls testen, welches Ereignis zuerst eintritt bzw. welche Ereignismethode – um die geht es ja schließlich – zuerst zur Ausführung gelangt. Das geht ganz einfach, indem Sie in den Block dieser Methoden (`{ ... }`) Anweisungen schreiben, die sich bei der Ausführung nach außen hin bemerkbar machen – am besten Ausgabeanweisungen. So nennt man Anweisungen, die für den Benutzer sichtbar einen Text ausgeben.

Die Anweisung `Console::WriteLine(L"Mein erstes Programm");`, die Sie in Ihrem Konsolenprogramm »Hallo-Welt« verwendet haben, ist beispielsweise eine solche Ausgabeanweisung.

Wenn Sie einmal im Zweifel sind, welches von mehreren Ereignissen zuerst eintritt, könnten Sie als Text für das `Click`-Ereignis etwa »Methode für Click« und als Text für das `Enter`-Ereignis »Methode für Enter« verwenden. Auch im Zeitalter aufwändiger Debug- und sonstiger Testwerkzeuge ergeben solche selbst geschriebenen Testszenarios immer noch Sinn. Sobald Sie sich dann über die Reihenfolge der Ereignisse im Klaren sind, löschen Sie die betreffenden Zeilen in Ihrem Code wieder.

Erweitern wir nun das, was wir über Schaltflächen gesagt haben, auf alle Steuerelemente. Steuerelemente sind Textfelder, Listenfelder, Schaltflächen, Fenster, Menüleisten usw. – kurz und gut: all das, was das Aussehen (die Benutzeroberfläche) einer Anwendung ausmacht. Also: Jedes Steuerelement besitzt eine Reihe von spezifischen Ereignissen mit zugehörigen Ereignismethoden.

Dass es diese Ereignisse überhaupt gibt, liegt daran, dass sie in den entsprechenden Klassen des .NET Framework definiert sind. Dass hinter jedem Steuerelement eine Klasse des .NET Framework steht, wissen Sie ja schon. Dabei können Sie sich gleich merken: Ohne Klassen geht im .NET Framework rein gar nichts. Alle Funktionalität, die Ihnen hier zur Verfügung steht, ist in Klassen gekapselt.

5.2 Steuerelemente mit Programmcode verbinden

Lassen Sie uns die Vorgehensweise gleich an einem kleinen Beispiel demonstrieren. Dazu werden wir das Hallo-Welt-Programm in der GUI-Version so erweitern, dass der Text des Beschriftungsfeldes seine Größe ändert, wenn ein Benutzer mit der Maus darüber fährt.

1. Öffnen Sie das Projekt *HalloWelt_GUI* in Visual C++ 2010 Express.

   Zur Erinnerung: Sie laden ein Projekt mit dem Menübefehl *Datei/Öffnen/Projekt/Projektmappe...* und Auswahl der Solution-Datei (Erweiterung *.sln*) in die Visual C++ Express-Oberfläche. Alternativ nutzen Sie eine Verknüpfung auf der Startseite (Abschnitt *Zuletzt geöffnete Projekte*).
2. Laden Sie die Entwurfsansicht der Datei *Form1.h* in den Arbeitsbereich, falls sie sich dort noch nicht befindet. Klicken Sie dazu im Projektmappen-Explorer einfach doppelt auf den Dateinamen.
3. Selektieren Sie im Windows Forms-Designer das Beschriftungsfeld.
4. Klicken Sie im Eigenschaftenfenster auf das Blitzsymbol (siehe Abbildung 5.1).

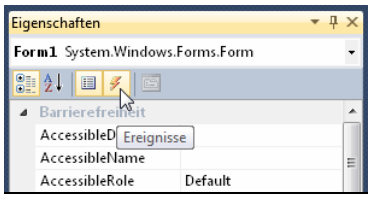

Abbildung 5.1: Über das Symbol mit dem Blitz wechseln Sie im Eigenschaftenfenster in die Ereignisansicht

Es erscheint eine Auflistung aller für das Label definierten Ereignisse. Sobald Sie die Einfügemarke auf ein bestimmtes Ereignis setzen, erhalten Sie unten im Eigenschaftenfenster in einem grau unterlegten Bereich eine Beschreibung, wann das Ereignis ausgelöst wird (siehe Abbildung 5.2; falls Sie den Bereich mit der Beschreibung nicht sehen, ziehen Sie ihn mit der Maus am unteren Rand auf).

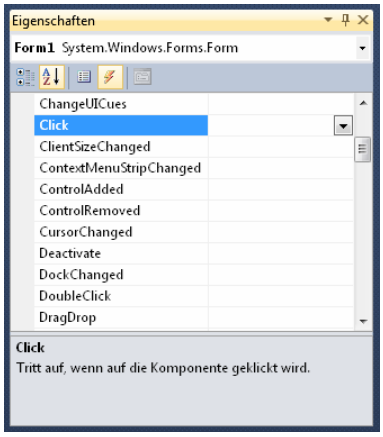

Abbildung 5.2: Ereignisse des Label-Steuerelements

Wie Sie Abbildung 5.2 entnehmen können, hat auch das Label-Steuerelement ein `Click`-Ereignis, das uns hier aber nicht weiter interessieren soll. Vielmehr wollen wir, dass sich die Schriftgröße des Hallo-Welt-Textes ändert, wenn der Benutzer diesen mit der Maus überstreicht. Natürlich könnten wir dieselbe Funktionalität auch für einen Mausklick einrichten.

Die einschlägigen Ereignisse für unser Vorhaben sind `MouseEnter` und `MouseLeave`. `MouseEnter` tritt ein, wenn die Maus über den sichtbaren Bereich des Labels bewegt wird und `MouseLeave`, wenn die Maus diesen Bereich wieder verlässt.

Sie müssen im Eigenschaftenfenster gegebenenfalls nach unten scrollen, um beide Ereignisse zu sehen. Die Ereignisse – für das Label-Steuerelement sind es über fünfzig – sind im Eigenschaftenfenster wie die Eigenschaften standardmäßig nach Kategorien aufgelistet. In diesem Fall finden Sie die Ereignisse *MouseEnter* und *MouseLeave* in einer separaten Kategorie für Mausereignisse *(Maus)*. Im Übrigen können Sie auch die Anzeige der Ereignisse über das Symbol mit dem *A* und dem *Z* in der oberen Symbolleiste alphabetisch ordnen lassen.

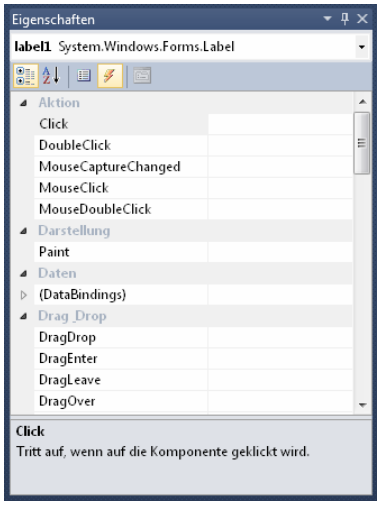

Abbildung 5.3: Ereignisse in der Kategorieansicht

5. Klicken Sie jetzt im Eigenschaftenfenster doppelt auf den Ereignisnamen *MouseEnter*.

   Daraufhin legt Visual C++ 2010 Express im Code eine leere Ereignismethode an und präsentiert Ihnen im Arbeitsbereich in einem neuen Fenster die Codeansicht. Der Cursor befindet sich im Rumpf der Methode, also an der Stelle, an der Sie gewöhnlich die erste Anweisung codieren werden.

```
private: System::Void label1_MouseEnter(System::Object^  sender, System::EventArgs^  e) {
             }
```

   Das Schlüsselwort `private` ist ein so genannter Zugriffsspezifizierer. Zugriffsspezifizierer regeln den Zugriff auf ein Element einer Klasse – zum Beispiel auf eine Methode – von außerhalb der Klasse. Über dieses Thema werden Sie mehr erfahren, wenn wir uns in den Kapiteln 10 und 11 mit den Konzepten der objektorientierten Programmierung beschäftigen.

   Ein Zugriffsspezifizierer gehört eigentlich nicht unmittelbar zu einer bestimmten Methode. Vielmehr gilt die Einstellung – hier `private` – für alle nachfolgenden Member der Klasse, solange im Code kein anderer Zugriffsspezifizierer auftritt. Aus diesem Grund ist die schließende geschweifte Klammer im Code auch eingerückt. Deshalb werden wir bis auf Weiteres auf den Abdruck von Zugriffsspezifizierern verzichten, wenn wir Methoden alleine, das heißt ohne äußeren Container, sprich: Klasse, darstellen:

```
System::Void label1_MouseEnter(System::Object^  sender, System::EventArgs^  e) {
}
```

   Der Name der generierten Ereignismethode setzt sich zusammen aus dem Namen des Steuerelements (hier *label1)* und – gefolgt von einem Unterstrich – dem Namen des Ereignisses. Er lautet hier also *label1_MouseEnter*. Über die Parameter der Methode und über die Bedeutung der Wörter `private`, `System` und `Void` brauchen Sie sich zum jetzigen Zeitpunkt noch keine Gedanken zu machen.

   Wenn Sie jetzt das Registerfenster, das den Code von *Form1.h* enthält, wieder schließen und danach im Eigenschaftenfenster erneut auf den Eintrag *MouseEnter* doppelklicken würden, würde Visual C++ 2010 Express natürlich keine zweite Ereignismethode anlegen, sondern Sie wieder zum Code der bereits bestehenden führen.

   Die Toolbox können Sie jetzt übrigens ausblenden, falls Sie dies nicht schon getan haben. Wir werden sie für dieses Beispiel nicht benötigen. Sie haben dann für das Codefenster mehr Platz zur Verfügung.

6. Notieren Sie im Rumpf der Ereignismethode *label1_MouseEnter()* – also zwischen den beiden geschweiften Klammern – die Anweisung `label1->Font = gcnew System::Drawing::Font("Arial", 23);`

```
System::Void label1_MouseEnter(System::Object^  sender, System::EventArgs^  e) {
   label1->Font = gcnew System::Drawing::Font("Arial", 23);
}
```

   Ob die öffnende geschweifte Klammer von Methoden in einer separaten Zeile oder, wie hier, in der Zeile des Methodenkopfs steht, ist im Übrigen eine reine Geschmacksfrage. Manche Programmierer bevorzugen für die Formatierung von Methoden diese Form, andere jene:

```
System::Void label1_MouseEnter(System::Object^  sender, System::EventArgs^  e)
{
   label1->Font = gcnew System::Drawing::Font("Arial", 23);
}
```

   Wir werden uns in diesem Buch beider Varianten bedienen. Im Sinne einer besseren Lesbarkeit des Codes sollten Sie jedoch in Ihren eigenen Programmen eine einmal gewählte Form grundsätzlich beibehalten.

Bevor wir weitergehen, eines noch vorweg: Verstehen Sie die nachfolgenden Erklärungen zum Code in diesem Kapitel als reine Zugabe. Hier geht es vor allem darum, Ihnen näherzubringen, wie Sie Steuerelemente mit Code verbinden, um auf Benutzeraktionen zu reagieren. Alles, was wir nun im Weiteren über die Syntax von C++ erwähnen, werden wir an anderer Stelle noch einmal ausführlicher behandeln.

Wenn wir jetzt schon auf einige Aspekte eingehen, dann geschieht das nach dem Motto: Es kann später nicht schaden, wenn Sie von dem einen oder anderen schon gehört haben. Sind Sie also nach der Lektüre dieses Abschnitts der Meinung, dass noch Fragen offen sind, vertrauen Sie einfach darauf, dass Sie die Antworten noch erhalten werden. Kehren Sie gegebenenfalls später noch einmal hierher zurück. Wahrscheinlich werden Sie dann feststellen, wie sich alles zusammengefügt hat.

Lassen Sie uns die Anweisung `label1->Font = gcnew System::Drawing::Font("Arial", 23);` in diesem Sinne einmal genauer unter die Lupe nehmen. Das abschließende Semikolon als Anweisungsende wird Ihnen jetzt sicher schon vertraut sein. Dabei handelt es sich übrigens um eine Eigenheit von C++, die bereits in der Sprache C ihren Ursprung hat[1] und die von anderen Programmiersprachen, welche ihre Wurzeln in C++ haben, übernommen wurde. Als Beispiele seien etwa Java, JavaScript, PHP, Perl und C# genannt.

Den ganz Aufmerksamen unter den Lesern wird vielleicht der qualifizierte Zugriff `System::Drawing::Font` ins Auge gefallen sein. Schließlich ist der Namensraum `System::Drawing` per `using`-Anweisung im Quelltext eingebunden – Sie finden die Zeile `using namespace System::Drawing;` oben im Code der Datei *Form1.h*.

In diesem Fall – wenn ein Namensraum mit `using` bekannt gemacht ist – ist ja eigentlich kein qualifizierter Zugriff notwendig. Dennoch würde die Anweisung `label1->Font = gcnew Font("Arial", 23);` hier eine Fehlermeldung hervorrufen.

Der Grund ist, dass die Formularklasse, in der wir uns befinden, selbst eine Eigenschaft `Font` besitzt, die den Bezeichner der gleichnamigen Klasse des Namensraums `Drawing` hier verdeckt. Vorrang hat so gesehen immer der Bereich, der am nächsten ist, und das ist hier natürlich der Bereich, der von der Klasse *Form1* umfasst wird, abgesteckt durch die öffnende und die schließende geschweifte Klammer.

```
public ref class Form1 : public System::Windows::Forms::Form
{
   ...
   ...
   private: System::Void label1_MouseEnter(System::Object^  sender, System::EventArgs^  e)
               {
                   label1->Font = gcnew System::Drawing::Font("Arial", 23);
                }
};
```

Tatsächlich spricht man in diesem Zusammenhang von Gültigkeitsbereichen und die öffnende Klammer ({) leitet deren Beginn ein, die schließende (}) deren Ende.

Auf die `Font`-Eigenschaft des Formulars können Sie innerhalb der Klasse also unqualifiziert zugreifen. Gleichzeitig verdeckt diese Eigenschaft andere Elemente gleichen Namens, sodass auf diese innerhalb der Klasse *Form1* nur der qualifizierte Zugriff möglich ist.

---

[1] Die Programmiersprache C++ ist eine reine Obermenge der Programmiersprache C. Das heißt, alles, was es in C gibt, gibt es in C++ auch. Aus dieser Tatsache leitet sich übrigens auch der Name C++ ab.

Wichtig: Am Ende einer Klasse steht immer ein Semikolon

Noch eine Eigenheit von C++, auf die wir Sie schon jetzt hinweisen wollen, weil man sie leicht vergisst: Die Definition von Klassen muss – anders als das bei Methoden oder auch bei Namensräumen der Fall ist – immer mit einem Semikolon hinter der schließenden geschweiften Klammer abgeschlossen werden.

Die beiden Namensräume `System` und `Drawing` sind übrigens ineinander verschachtelt. Sie können sich das etwa so vorstellen:

```
namespace System
{
   ...
   namespace Drawing
   {
      ...
   }
   ...
}
```

Wenn Sie die Klassen eines Namensraums im Code direkt, das heißt ohne qualifizierten Zugriff, verwenden wollen, müssen Sie diesen Namensraum immer explizit einbinden. Es reicht also nicht aus, `System` einzubinden, wenn Sie Klassen in `Drawing` verwenden wollen. Mit der Direktive

```
using namespace System;
```

machen Sie nur den Namensraum `System` mit seinen Klassen bekannt, nicht aber darin geschachtelte andere Namensräume. Erforderlich ist also

```
using namespace System::Drawing;
```

Wenn Sie Klassen in beiden Namensräumen verwenden wollen, müssen Sie dementsprechend auch beide `using`-Anweisungen notieren.

Kommen wir wieder auf die Anweisung zurück, die wir in der Ereignismethode eingerichtet haben. Etwa in ihrer Mitte steht das Zeichen (`=`). Sie kennen es aus der Mathematik als Gleichheitszeichen. Nun, in der Programmierung stellt dieses Zeichen einen so genannten Zuweisungsoperator dar. Über diesen werden Sie im nächsten Kapitel noch eine ganze Menge zu lesen bekommen.

Hier schon mal soviel: Die Bedeutung des Zuweisungsoperators ist eine ganz andere, als Sie es von der Mathematik her kennen. Wie der Name schon erahnen lässt, weist er etwas zu, nämlich einen Wert. Dieser Wert steht immer rechts vom Zuweisungsopertor. Hier handelt es sich um den Ausdruck `gcnew System::Drawing::Font("Arial", 23);`.

Als Ausdruck bezeichnet man in der Programmierung alles, was einen Wert repräsentiert. So stellt der Ausdruck `gcnew System::Drawing::Font("Arial", 23);` ein `Font`-Objekt dar, dessen Eigenschaften ein genau definiertes Schriftbild festlegen. Das Objekt wird an dieser Stelle erst erzeugt, nämlich mit `gcnew`.

Mit der Zuweisung erhält die `Font`-Eigenschaft des Label-Steuerelements (`label1->Font`) schließlich diesen Wert. Eigentlich handelt es sich hier ja um mehrere Werte, die den Font bestimmen. Es ist jedoch so, dass Objekte von Klassen verschiedene Werte praktisch zu einer Einheit zusammenfassen. Von daher kann man sie auch wie einen einzigen Wert behandeln – das gilt zumindest im Kontext einer Zuweisung.

Hinweis: C++/CLI-Code

Wenn Sie im .NET Framework Objekte erzeugen, machen Sie das immer mit `gcnew`. Die Syntax weicht etwas vom C++-Standard ab (dort verwendet man den Operator `new`, ohne »gc«[2]).

Ebenso erfolgt im Standard-C++ der Zugriff auf Eigenschaften und Methoden von Klassen über den Punktoperator in der Form Objektname.Eigenschaft und nicht, wie im .NET Framework, mit dem Pfeiloperator (Objektname->Eigenschaft wie zum Beispiel in `label1->Font`).

Die Syntax, die im .NET Framework verwendet wird, hat auch einen besonderen Namen, nämlich C++/CLI. Im Übrigen sind die Unterschiede nicht besonders groß. C++-Programmierer, die bis jetzt nicht für .NET Framework entwickelt haben, werden sich schnell an die C++/CLI-Syntax gewöhnen.

Merken Sie sich am besten schon jetzt, dass im Zuge einer Zuweisung, dargestellt durch den Operator (=), auf der rechten Seite immer ein Wert steht bzw. ein Ausdruck, der einen solchen repräsentiert, und auf der linken Seite ein Datenobjekt, das diesen Wert zugewiesen bekommt. Wie gesagt, mehr dazu erfahren Sie im nächsten Kapitel.

Nachdem wir nun der Eigenschaft `Font` im Code der Ereignismethode *label1_MouseEnter()* einen neuen Wert zugewiesen haben, nämlich die Schriftgröße 23 (wir hatten sie zuvor ja im Windows Forms-Designer über das Eigenschaftenfenster mit der Größe 33 festgelegt), wird sich diese Änderung bemerkbar machen, sobald ein Benutzer bei der Programmausführung das Ereignis `MouseEnter` auslöst. Probieren Sie es gleich einmal aus.

7. Starten Sie die Anwendung in Visual C++ 2010 Express mit F5. Wenn das Anwendungsfenster erscheint, fahren Sie mit der Maus über den Text »Hallo Welt«. Die Schrift verkleinert sich, sobald die Maus in den Bereich des Beschriftungsfeldes gelangt (siehe Abbildung 5.4).

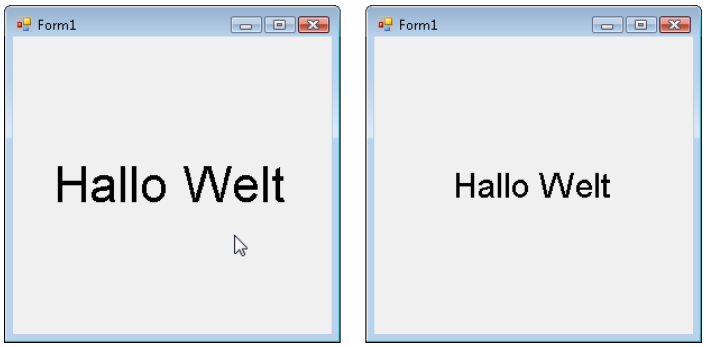

Abbildung 5.4: Vorher/Nachher: Der Begrüßungstext wird kleiner, wenn ein Benutzer mit der Maus darüber fährt

Allerdings bleibt die Schrift danach so, wie sie ist. Wir wollen aber erreichen, dass sie wieder ihre ursprüngliche Größe annimmt, nachdem sich die Maus entfernt hat.

[2] Das »gc« steht für Garbage Collection. Es handelt sich um die Speicherverwaltung des .NET Framework, über die Sie noch einiges hören werden. Sie funktioniert praktisch automatisch, ohne dass sich der C++-Programmierer darum kümmern müsste. (Das Nichtaufräumen von Speicherressourcen ist von jeher eine häufige Fehlerquelle in unmanaged C++-Programmen – unmanaged heißt: nicht vom .NET Framework verwaltet.)

Dazu bemühen wir ein zweites Ereignis, das wir oben schon genannt haben: `MouseLeave`. Es tritt ein, wenn die Maus den Bereich des Labels verlässt, also genau zu dem Zeitpunkt, den wir für unsere Zwecke benötigen.

Zunächst erzeugen Sie auf die inzwischen bekannte Weise den zum `MouseLeave`-Ereignis korrespondierenden Eventhandler. In diesem setzen Sie den Wert für die Schriftgröße dann wieder auf die ursprünglichen 33 Punkte.

1. Wechseln Sie in die Entwurfsansicht Ihres Formulars.
2. Selektieren Sie im Arbeitsbereich das Label-Steuerelement.
3. Klicken Sie in der Ereignis-Ansicht des Eigenschaftenfensters doppelt auf den Eintrag für das `MouseLeave`-Ereignis.

   Danach wechselt Visual C++ 2010 Express wieder in die Codeansicht und bietet Ihnen das Grundgerüst der passenden Ereignismethode an.

```
private: System::Void label1_MouseLeave(System::Object^  sender, System::EventArgs^  e){
}
```

4. Notieren Sie in der Methode *label1_MouseLeave()* die Anweisung `label1->Font = gcnew System::Drawing::Font("Arial", 33);`.

```
private: System::Void label1_MouseLeave(System::Object^  sender, System::EventArgs^  e)
{
   label1->Font = gcnew System::Drawing::Font("Arial", 33);
}
```

5. Führen Sie das Programm aus. Bewegen Sie die Maus auf den Begrüßungstext und dann wieder von ihm weg. Nun verkleinert sich der Text in bekannter Weise, wenn Sie mit der Maus darüber fahren und nimmt wieder seine ursprüngliche Größe an, sobald Sie den Mauszeiger entfernen. Sie können das mit dem gleichen Effekt beliebig oft wiederholen.

Die Eigenschaft Anchor

Eine Kleinigkeit sollten wir noch korrigieren. Wenn Sie bei laufendem Programm das Anwendungsfenster maximieren, werden Sie feststellen, dass der Text »Hallo Welt« nicht zentriert bleibt. Der gleiche Effekt stellt sich ein, wenn Sie die Größe des Fensters durch Ziehen mit der Maus verändern.

Verantwortlich für dieses Verhalten ist die `Anchor`-Eigenschaft des Label-Steuerelements. Dazu sei gesagt, dass nicht nur Beschriftungsfelder, sondern auch Schaltflächen und eine ganze Reihe anderer Steuerelemente diese Eigenschaft besitzen.

Die `Anchor`-Eigenschaft legt fest, an welche Ränder des übergeordneten Containers ein Steuerelement gekoppelt ist. Wobei hier als übergeordneter Container des Label-Steuerelements das Formular zu verstehen ist.

Standardmäßig steht die `Anchor`-Eigenschaft auf `Top` und `Left`. Das heißt, wenn das Fenster seine Größe ändert, bleibt der Abstand des Labels zum oberen und zum linken Rand konstant.

Um das zu ändern, haben Sie zwei Möglichkeiten: Entweder Sie setzen – praktisch als Ausgleich – die Werte auch für die gegenüberliegenden Seiten, also sowohl für `Top` und `Left` als auch für `Right` und `Bottom`. Einfacher und logischer ist es jedoch, die `Anchor`-Eigenschaft auf den Wert `None` zu setzen. Schließlich soll das Beschriftungsfeld ja an überhaupt keinen Rand gekoppelt sein. Wie gesagt, der Effekt ist der gleiche.

1. Selektieren Sie das Label-Steuerelement in der Entwurfsansicht.
2. Wechseln Sie im Eigenschaftenfenster in die Eigenschaftenansicht.
3. Selektieren Sie dort die `Anchor`-Eigenschaft des Labels.

   Nun können Sie es sich aussuchen: Entweder Sie tippen den Wert für die `Anchor`-Eigenschaft in der Spalte daneben von Hand ein. Für unsere Zwecke ist das `None`, was bedeutet, dass das aktuelle Steuerelement an keinen Rand des übergeordneten Containers gekoppelt ist. Alternativ benutzen Sie das nachfolgend beschriebene Dialogfeld, um den gewünschten Wert einzustellen. Wir bevorzugen hier die zweite Möglichkeit, damit Sie diese einmal kennen lernen.
4. Klicken Sie in der Spalte daneben auf die Schaltfläche mit dem nach unten gerichteten Dreieck.

   Es erscheint ein kleines Fenster, in dem die Abstände vom aktuellen Steuerelement zu aktivierten Rändern – gemeint sind diejenigen Ränder, an die das Steuerelement gekoppelt ist – mit grauen Balken dargestellt sind (siehe Abbildung 5.5).

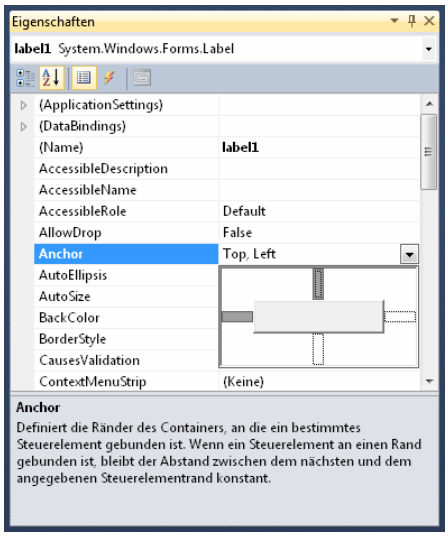

Abbildung 5.5: Bearbeiten der Anchor-Eigenschaft im Eigenschaftenfenster

5. Klicken Sie die grauen Balken jeweils einmal an und entfernen Sie anschließend den Fokus.

   Sie brauchen den Fokus nur von dem kleinen Dialogfeld zu entfernen. Dafür reicht es schon aus, wenn Sie den Cursor in eine der beiden Spalten der `Anchor`-Eigenschaft zurücksetzen. Daraufhin stellt sich der Wert `None` in der Spalte neben *Anchor* automatisch ein.

   Wie gesagt, wenn Sie alle Balken grau einfärben, indem Sie die weißen je einmal anklicken, erzielen Sie den gleichen Effekt. Die Werte heben sich dann praktisch gegenseitig auf.
6. Führen Sie das Programm erneut aus und maximieren Sie das Anwendungsfenster bzw. vergrößern und/oder verkleinern Sie es durch Ziehen mit der Maus.

   Der Beschriftungstext bleibt nunmehr auch dann zentriert, wenn sich die Größe des Anwendungsfensters ändert.

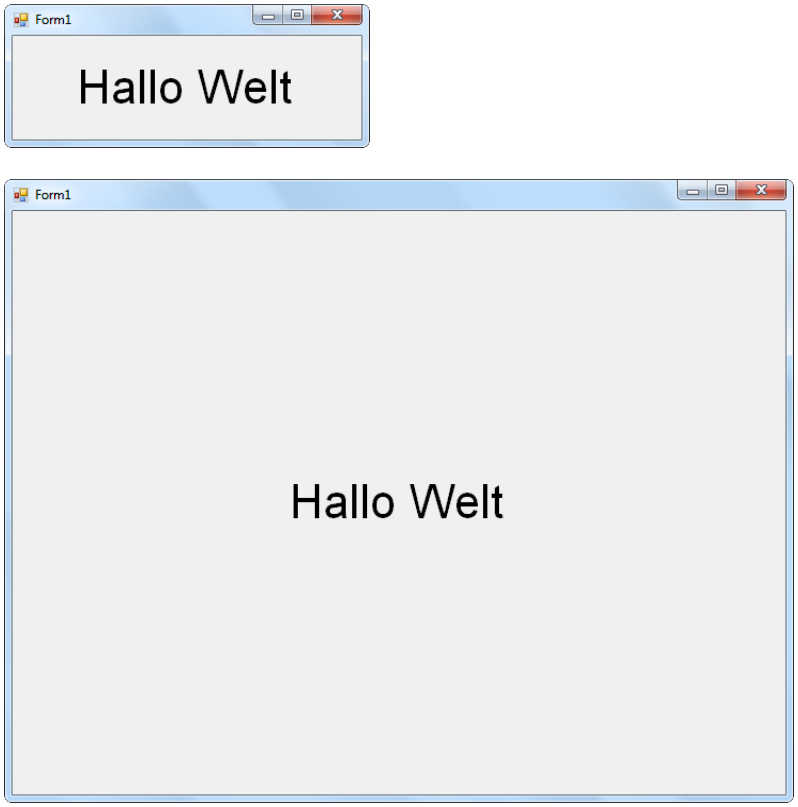

Abbildung 5.6: Mit dem Wert `None` für die `Anchor`-Eigenschaft bleibt das Label auch dann zentriert, wenn das Anwendungsfenster seine Größe ändert

5.3 Dialogfenster integrieren

Wenn Sie weiter fortgeschritten sind und Ihre Programme umfangreicher werden, werden Sie oft zusätzliche Dialogfenster benötigen. Denken Sie zum Beispiel an die Dialogfelder zum Öffnen und Speichern von Dokumenten oder an das Dialogfeld, mit dem der Benutzer Einstellungen vor dem Ausdrucken von Dokumenten vornehmen kann. Für Microsoft Office und viele andere Programme sind solche Dialogfelder Standard. Der Programmierer realisiert sie mit zusätzlichen Fenstern, die über das Hauptfenster der Anwendung gestartet werden.

Da wir gerade so viel über das Anwendungsfenster gesprochen haben und somit praktisch beim Thema sind, wollen wir Ihnen in einer kleinen Übung auch gleich zeigen, was Sie tun müssen, um zusätzliche Dialogfenster zu erzeugen. Im Wesentlichen sind dazu drei Schritte notwendig:

- Sie instanziieren im Code ein Fensterobjekt
- Sie konfigurieren das Fenster mit den Eigenschaften dieses Objekts
- Sie zeigen das Fenster mit einer der beiden Methoden `Show()` oder `DialogShow()` an

Im Beispiel werden wir das Hallo-Welt-Fenster über eine Schaltfläche des Hauptfensters anzeigen.

Hinweis: Projektnamen der Beispiele

Damit Sie sich beim Suchen der Beispiele besser zurecht finden, werden wir kleinere Beispielprojekte des Öfteren nach der Kapitelnummer plus eines fortlaufenden Buchstabens für die Reihenfolge benennen, also *K<Kapitelnummer>a*, *K<Kapitelnummer>b*, *K<Kapitelnummer>c* usw.

Wenn ein Beispielprojekt einen ausführlichen Namen trägt, überspringen wir bei der Namensgebung für das nächste Beispiel den entsprechenden Buchstaben. So heißen die Beispielprojekte für Kapitel 5, die Sie im Ordner *Beispiele/K05* finden, *HalloWelt_GUI* und *K05b*. Der Name *HalloWelt_GUI* ersetzt dabei den Namen *K05a*.

1. Legen Sie in Ihrer Visual C++ 2010 Express-IDE ein neues Projekt mit der Projektvorlage *Windows Forms-Anwendung* an.
2. Ziehen Sie von der Toolbox ein Button-Steuerelement auf Ihr Formular. Sie finden das Steuerelement *Button* ebenfalls in der Kategorie *Allgemeine Steuerelemente* (oder unter *Alle Windows Forms*).

   Der Name des neuen Button-Steuerelements ist *button1*. Sie sehen den Namen im Arbeitsbereich auf dem Steuerelement oder im Eigenschaftenfenster (Eigenschaft *(Name)*), wenn die Schaltfläche auf dem Formular selektiert ist. Sie können den Namen auch ändern, indem Sie im Eigenschaftenfenster den Wert für diese Eigenschaft neu setzen. Wenn Sie das wollen, sollten Sie das jetzt sofort tun.

   Als Nächstes konfigurieren Sie die Schaltfläche.
3. Überzeugen Sie sich davon, dass das Button-Steuerelement auf dem Formular selektiert ist und begeben Sie sich ins Eigenschaftenfenster.
4. Ändern Sie den Wert der Eigenschaft `Text` zu **KLICK MICH**.

   Damit legen Sie die Beschriftung der Schaltfläche fest. Selbstverständlich können Sie auch jeden anderen Text verwenden, der Ihnen gefällt.
5. Selektieren Sie im Eigenschaftenfenster die `AutoSize`-Eigenschaft.
6. Klicken Sie mit der Maus in das Feld daneben und stellen Sie über die erscheinende Schaltfläche mit dem kleinen Dreieck den Wert für `AutoSize` auf `True` ein.

   Damit bestimmen Sie, dass sich die Größe der Schaltfläche dem enthaltenen Text automatisch anpasst. Das ist sehr praktisch, da es Ihnen hier erspart, die Konfiguration der Eigenschaft `Size` mit den Werten für `Width` und `Height` vorzunehmen.
7. Setzen Sie die `Anchor`-Eigenschaft der Schaltfläche auf `None`.

   Den Font bearbeiten Sie wiederum im Dialog *Schriftart*, zu erreichen über die kleine Schaltfläche mit den drei Punkten in der Spalte neben *Font* (siehe auch Abbildung 4.8 des letzten Kapitels). Die Schaltfläche erscheint, wenn Sie im Eigenschaftenfenster die `Font`-Eigenschaft selektiert haben.
8. Stellen Sie als Schriftart *Arial*, für den Schriftschnitt *Standard* und für den Schriftgrad *26* ein.

   Lassen Sie uns nun zur Abwechslung noch die Hintergrund- und die Schriftfarbe der Schaltfläche explizit festlegen.
9. Stellen Sie für die Eigenschaften `ForeColor` (Schriftfarbe) und `BackColor` (Hintergrundfarbe) je eine Farbe Ihrer Wahl ein.

Wählen Sie eine der vordefinierten Farben in den Registern *System* und *Web* oder suchen Sie sich im Register *Benutzerdefiniert* eine Farbe aus. Im Beispiel haben wir für den Hintergrund ein helles Grau (RGB-Wert: *224; 224; 224*) und für die Schrift blau (Wert: *Blue*) gewählt.

Die Werte für Rot, Grün und Blau können Sie auch direkt im RGB-Format von Hand in das Feld eintragen. Alternativ können Sie auch vordefinierte Konstanten für Farbnamen wie *Blue*, *Red*, *Yellow* etc. verwenden.

Nun vergrößern Sie das Formular noch etwas.

10. Selektieren Sie das Formular im Windows Forms-Designer.
11. Ziehen Sie das Formular mit der Maus an einem der Griffe, bis es die von Ihnen gewünschte Größe besitzt.

    Wenn Sie rechts bzw. unten ziehen, ändert sich das Formular nur in der Breite bzw. in der Höhe. Wenn Sie, wie in Abbildung 5.7 zu sehen, am Eckpunkt ziehen, ändern Sie Breite und Höhe, wobei das Seitenverhältnis erhalten bleibt. Auf die gleiche Weise können Sie im Übrigen auch die meisten anderen Steuerelemente in der Größe anpassen. Daneben bleibt natürlich immer noch die Bearbeitung der *Size*-Eigenschaft im Eigenschaftenfenster.

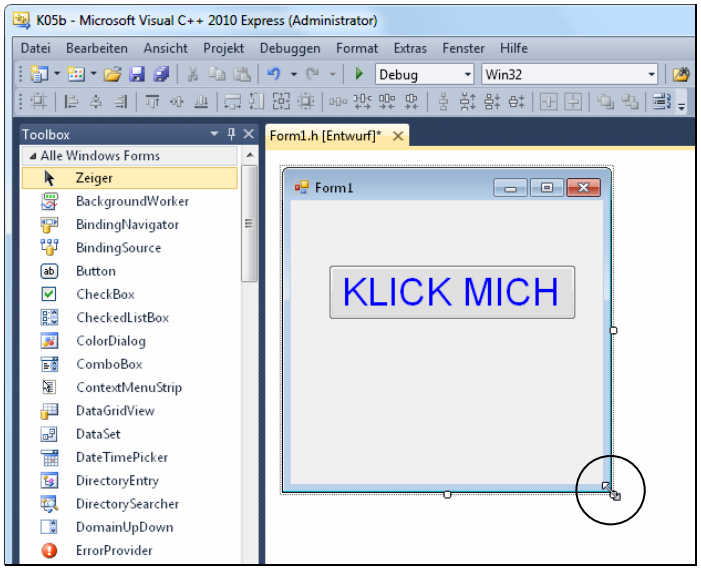

Abbildung 5.7: Die Größe vieler Steuerelemente – wie hier die des Formulars – können Sie durch Ziehen mit der Maus anpassen

Im Beispielprojekt hat das Formular den Wert *350* für die Breite und für die Höhe. Denken Sie bitte daran, das Formular vorher im Windows Forms-Designer zu selektieren, falls Sie die Werte im Eigenschaftenfenster einstellen wollen. Klicken Sie dazu im Formular auf eine beliebige freie Stelle des Formulars – also außerhalb der Schaltfläche.

Schließlich wählen wir noch den Text »Dialogfenster anzeigen« für die Titelleiste des Formulars.

12. Selektieren Sie die `Text`-Eigenschaft des Formulars im Eigenschaftenfenster.
13. Löschen Sie in der zweiten Spalte den Text »Form1« und ersetzen Sie ihn durch den Text **Dialogfenster anzeigen**.

Zu guter Letzt zentrieren Sie noch die Schaltfläche auf dem Formular.

14. Selektieren Sie das `Button`-Steuerelement in der Entwurfsansicht.
15. Verwenden Sie im Menü *Format* nacheinander die Befehle *Auf Formular zentrieren/Horizontal* und *Auf Formular zentrieren/Vertikal*.

Abbildung 5.8: Anwendungsfenster mit Schaltfläche

Damit steht Ihre Schaltfläche in ansprechendem Design da. Nun gilt es, die gewünschte Funktionalität für die Schaltfläche zu implementieren. Wie das geht, das wissen Sie ja schon: über Ereignisse bzw. über eine passende Ereignismethode.

Standardereignisse

Natürlich soll das Dialogfenster auf einen Schaltflächenklick hin in Erscheinung treten. Über das zuständige Ereignis haben wir ja schon gesprochen, es heißt `Click`.

Nun könnten Sie wie im letzten Beispiel beim Beschriftungsfeld vorgehen, um eine Ereignismethode zu erstellen; also die Schaltfläche in der Entwurfsansicht markieren und in der Ereignisansicht des Eigenschaftenfensters den Namen des Ereignisses doppelt anklicken. Für so genannte Standardereignisse – zu denen das `Click`-Ereignis des Button-Steuerelements gehört – geht es jedoch auch einfacher.

1. Klicken Sie im Windows Forms-Designer doppelt auf das Button-Steuerelement. Daraufhin setzt Visual C++ 2010 Express das Methodengerüst der Ereignismethode *button1_Click()* für Sie auf und wechselt in die Codeansicht.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
}
```

Zur Erklärung: Jedes Steuerelement besitzt ein Standardereignis. Für Schaltflächen ist das Standardereignis das `Click`-Ereignis, für Textfelder ist es zum Beispiel das `TextChanged`-Ereignis – dieses tritt ein, sobald sich der Wert eines Textfeldes ändert.

Um für Standardereignisse eine leere Ereignismethode[3] anzulegen, genügt der gerade erwähnte Doppelklick auf das jeweilige Steuerelement. Für die anderen Ereignisse (oder alternativ auch für Standardereignisse) legen Sie die entsprechenden Methoden in bekannter Weise an, indem Sie den Ereignisnamen im Eigenschaftenfenster doppelt anklicken.

Tabelle 5.1 zeigt eine Übersicht der Standardereignisse für häufig verwendete Steuerelemente nebst Beschreibung. Die Steuerelemente `Label` und `Button` kennen Sie bereits. Die anderen Steuerelemente, die in der Tabelle genannt sind, stellen wir Ihnen im nächsten Abschnitt kurz vor.

Tabelle 5.1: Wichtige Steuerelemente und die dazugehörigen Standardereignisse

Steuerelement	Standardereignis	Beschreibung
Label	Click	Tritt ein, wenn ein Benutzer das Label anklickt.
Button	Click	Tritt ein, wenn ein Benutzer die Schaltfläche anklickt.
TextBox	TextChanged	Tritt ein, wenn sich der Text in der TextBox ändert.
RichTextBox	TextChanged	Tritt ein, wenn sich der Text in der RichTextBox ändert.
CheckBox	CheckedChanged	Tritt ein, wenn sich der Wert der Checked-Eigenschaft ändert.
RadioButton	CheckedChanged	Tritt ein, wenn sich der Wert der Checked-Eigenschaft ändert.
MenuStrip	ItemClicked	Tritt ein, wenn ein Benutzer auf einen Menüeintrag klickt.
ToolTip	Popup	Tritt unmittelbar vor dem Anzeigen der Quickinfo ein.

Nun, für zusätzliche Dialogfenster steht Ihnen der Windows Forms-Designer nicht zur Verfügung. Den Code dafür müssen Sie also selbst schreiben. Als Erstes erzeugen Sie ein Objekt der Fensterklasse *Form*.

2. Tippen Sie im Rumpf der Ereignismethode *button1_Click()* die Anweisung `Form ^f = gcnew Form();` ein.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
   Form ^f = gcnew Form();
}
```

   Mit der Anweisung `Form ^f = gcnew Form();` erzeugen Sie ein Objekt der Klasse `Form`, das Sie im weiteren Code mit dem Bezeichner *f* ansprechen können. (Für das Zeichen mit dem Hütchen (`^`) müssen Sie die entsprechende Taste und anschließend die Leertaste drücken[4].) Wenn man im Code auf diese Weise ein Objekt (eine Instanz) einer Klasse erstellt, sagt man auch, man instanziiert[5] das Objekt.

Damit haben Sie auch schon ein Formular erstellt. Es entspricht im Aussehen dem Formular, das Ihnen der Windows Forms-Designer nach dem Anlegen einer neuen Windows Forms-Anwendung im Arbeitsbereich präsentiert.

[3] Von leeren Methoden spricht man, wenn diese keine Anweisungen enthalten, also nur aus dem Methodenkopf und den beiden geschweiften Klammern für den Methodenrumpf bestehen.

[4] Dieses Zeichen gehört ebenfalls zur C++/CLI-Syntax.

[5] Oft ist auch die Schreibweise »instanziert« zu finden.

Modale und nicht-modale Dialoge

Um Dialogfenster anzuzeigen, stehen Ihnen grundsätzlich zwei Methoden zur Verfügung: `Show()` und `ShowDialog()`. Mit der Methode `Show()` öffnen Sie ein nicht-modales Fenster, mit der Methode `ShowDialog()` ein modales.

Beide Dialogformen sind Ihnen sicher aus zahlreichen Anwendungen bekannt. Modale Dialogfelder behalten den Fokus so lange, bis sie geschlossen werden. Dem Benutzer ist es zwischenzeitlich nicht möglich, zum Beispiel zum Hauptfenster zurückzukehren. Ein typisches Beispiel hierfür sind Meldungsfenster. Solange sie nicht mit *OK* bestätigt werden, verharrt die Programmausführung im Status quo.

Zwischen nicht-modalen Fenstern dagegen kann der Benutzer wechseln, wann immer er will. Der Programmlauf wird nicht angehalten. Viele Dialogfelder in Office-Programmen sind nicht-modal, das *Suchen und Ersetzen*-Dialogfeld gehört beispielsweise auch dazu. (Dagegen sind das *Drucken*- und das *Speichern unter*-Dialogfeld wiederum modal – probieren Sie es einfach einmal aus.)

Wir entscheiden uns hier für ein modales Fenster, verwenden also die Methode `ShowDialog()`. Sie verhindern damit, dass der Benutzer zum Hauptformular zurückkehren und dort über die Schaltfläche neue Dialogfenster öffnen kann, solange das aktuelle nicht geschlossen ist.

1. Fügen Sie im Code der *button1_Click()*-Methode die Anweisung `f->ShowDialog();` hinzu.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
   Form ^f = gcnew Form();
   f->ShowDialog();
}
```

2. Schauen Sie sich das Ergebnis gleich einmal an und starten Sie Ihre Anwendung.
3. Klicken Sie im Anwendungsfenster auf die Schaltfläche mit der Aufschrift *KLICK MICH*. Jetzt sollte das Dialogfenster erscheinen.
4. Versuchen Sie bei geöffnetem Dialogfenster den Cursor in das Hauptfenster zu setzen.

   Der Versuch sollte nicht gelingen, da es sich eben um ein modales Dialogfenster handelt.

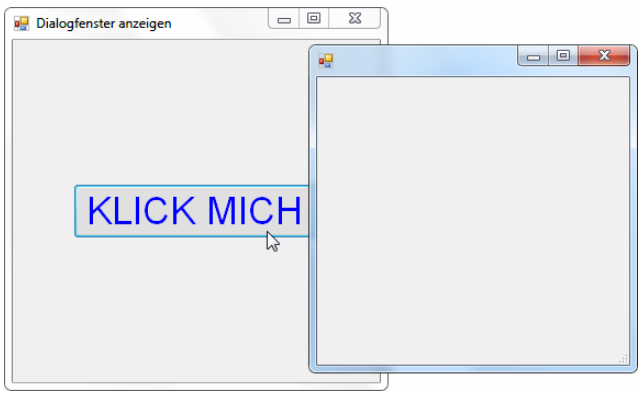

Abbildung 5.9: Der Benutzer erreicht die Schaltfläche nicht. Das Hauptfenster bleibt so lange gesperrt, bis das modale Dialogfenster geschlossen wird

5. Schließen Sie zuerst das Dialogfenster und danach das Hauptfenster.

Was Sie jetzt noch zu tun haben, ist das Dialogfenster zu konfigurieren. Zunächst stellen Sie für dieses über die Eigenschaften `Width` und `Height` die gewünschte Größe ein. Auf diese Eigenschaften greifen Sie mit dem Pfeiloperator über die eben erzeugte Instanz der `Form`-Klasse zu, der Sie den Namen – sprich: Bezeichner – *f* gegeben haben. Das `^`-Symbol geben Sie dabei nicht mit an.

1. Tippen Sie die beiden Anweisungen in der Ereignismethode *button1_Click()* hinter `Form ^f = gcnew Form()` ein.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
   Form ^f = gcnew Form();
   f->Width = 300;
   f->Height = 300;
   f->ShowDialog();
}
```

   Es liegt auf der Hand, dass Sie eine Instanz zuerst erzeugen müssen, bevor Sie auf deren Eigenschaften zugreifen können. Aus diesem Grund wäre es natürlich fehlerhaft, wenn Sie die Anweisungen `f->Width = 300;` und `f->Height = 300;` im Code der *button1_Click()*-Methode vor der Zeile `Form ^f = gcnew Form();` notieren würden.

   Als Nächstes fügen Sie Ihrem Dialogfenster ein Beschriftungsfeld hinzu. Das heißt, Sie benötigen erst einmal ein Objekt der Klasse `Label`. Dieses erzeugen Sie in inzwischen bekannter Weise mit dem `gcnew`-Operator.

2. Ergänzen Sie die Methode *button1_Click()* um die Anweisung `Label ^l = gcnew Label();`.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
   Form ^f = gcnew Form();
   f->Width = 300;
   f->Height = 300;
   Label ^l = gcnew Label();
   f->ShowDialog();
}
```

   Der Bezeichner des Labels ist jetzt *l*. Natürlich können Sie auch einen Namen wählen, der Ihnen besser gefällt, zum Beispiel *meinLabel* oder *fensterLabel* usw. In diesem Fall benutzen Sie diesen Bezeichner bei der Instanzierung:

```
Label ^meinLabel = gcnew Label();
```

   beziehungsweise

```
Label ^fensterLabel = gcnew Label();
```

   Im Weiteren verwenden Sie für den Zugriff auf die Eigenschaften Ihres Labels dann natürlich den Bezeichner, den Sie bei der Instanzierung gewählt haben (z.B. `meinLabel->Text = "Hallo Welt";`).

   Berücksichtigen Sie dabei aber, dass C++ zwischen Groß- und Kleinschreibung unterscheidet. Wenn Sie beim Instanzieren eines Objekts den Bezeichner *meinLabel* gewählt haben, dann müssen Sie diesen im weiteren Code auch so schreiben und nicht etwa *meinlabel*, *MeinLabel* oder *Meinlabel*. Letztere wären Namen, mit denen Ihr Compiler im Kontext nichts anfangen könnte.

Hinweis: camelCasing

Im .NET Framework hat es sich eingebürgert, die Bezeichner von Objekten mit einem Kleinbuchstaben beginnen zu lassen. Setzt sich ein Bezeichner aus mehreren Wörtern zusammen, dann wird jedes weitere Wort groß geschrieben. Hier einige Beispiele: *meinHaus, bigPoint, teilnehmer* (setzt sich nicht aus mehreren Wörtern zusammen), *anzahlTeilnehmer, anzahl_Schiffe*. (Der Unterstrich ist – abgesehen von den Umlauten – als einziges Sonderzeichen in Bezeichnern erlaubt.) Die genannte Schreibweise hat übrigens auch einen Namen: man nennt sie camelCasing.

Über den Bezeichner *l*, den wir hier gewählt haben, konfigurieren wir jetzt erst einmal das Beschriftungsfeld, bevor wir dieses dem Dialogfenster *f* hinzufügen. Als Beschriftungstext wählen wir wieder »Hallo Welt«. Den Beschriftungstext eines Labels legen Sie, wie Sie wissen, über dessen `Text`-Eigenschaft fest.

3. Weisen Sie im Code der `Text`-Eigenschaft des eben instanzierten Labels die Zeichenkette `"Hallo Welt"` zu.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
   Form ^f = gcnew Form();
   f->Width = 300;
   f->Height = 300;
   Label ^l = gcnew Label();
   l->Text = "Hallo Welt";
   f->ShowDialog();
}
```

4. Setzen Sie die `AutoSize`-Eigenschaft des Labels auf `true`.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
   Form ^f = gcnew Form();
   f->Width = 300;
   f->Height = 300;
   Label ^l = gcnew Label();
   l->Text = "Hallo Welt";
   l->AutoSize = true;
   f->ShowDialog();
}
```

   Nun passt sich die Größe des Labels automatisch dem Beschriftungstext an.

Die Schlüsselwörter `true` und `false` – beide klein geschrieben – repräsentieren so genannte boolesche Werte. Diese nennt man nach ihrem Datentyp *Boolean*. Boolesche Werte sind Wahrheitswerte – `true` steht für wahr, `false` für falsch bzw. unwahr. Man verwendet boolesche Werte vor allem dann, wenn es nur zwei Möglichkeiten gibt. So haben es auch die Entwickler des .NET Framework mit der Eigenschaft `AutoSize` gehalten: Entweder diese Eigenschaft ist aktiviert (Wert `true`) oder sie ist nicht aktiviert (Wert `false`).

Die nämlichen Werte sind im Eigenschaftenfenster groß geschrieben: *True* bzw. *False*. Im Code müssen Sie sie aber auf jeden Fall klein schreiben, da C++ case-sensitiv ist, heißt: zwischen Groß- und Kleinschreibung unterscheidet.

Mit den Eigenschaften `Left` und `Top` bestimmen Sie, wo das Label später auf dem Dialogfenster angezeigt wird. Der Wert für die Eigenschaft `Left` bezeichnet den Abstand zum linken Fensterrand, der Wert für die Eigenschaft `Top` den Abstand zum oberen Fensterrand.

5. Weisen Sie den Eigenschaften `Left` und `Top` des Labels die Werte *35* und *105* zu.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
   Form ^f = gcnew Form();
   f->Width = 300;
   f->Height = 300;
   Label ^l = gcnew Label();
   l->Text = "Hallo Welt";
   l->AutoSize = true;
   l->Left = 35;
   l->Top = 105;
   f->ShowDialog();
}
```

Für den Text wählen wir die Schriftart Arial mit der Schriftgröße 33. Im Code benötigen Sie dazu ein `Font`-Objekt, das Sie der gleichnamigen Eigenschaft des Labels zuweisen. Die Klasse `Font` ist im Namensbereich `Sytem::Drawing` definiert, den Sie, da die Eigenschaft des Label-Steuerelements den gleichen Namen trägt, zur Unterscheidung beim Zugriff mit angeben müssen.

6. Ergänzen Sie den Code der *button1_Click()*-Methode um die Anweisung `l->Font = gcnew System::Drawing::Font("Arial", 33);`.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
   Form ^f = gcnew Form();
   f->Width = 300;
   f->Height = 300;
   Label ^l = gcnew Label();
   l->Text = "Hallo Welt";
   l->AutoSize = true;
   l->Left = 35;
   l->Top = 105;
   l->Font = gcnew System::Drawing::Font("Arial", 33);
   f->ShowDialog();
}
```

Jetzt sollten Sie noch die `TextAlign`-Eigenschaft des Label-Steuerelements auf `MiddleCenter` und die `Anchor`-Eigenschaft auf `None` setzen.

7. Fügen Sie im Code der *button1_Click()*-Methode die Anweisungen `l->TextAlign = ContentAlignment::MiddleCenter;` und `l->Anchor = AnchorStyles::None;` hinzu.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
   Form ^f = gcnew Form();
   f->Width = 300;
   f->Height = 300;
   Label ^l = gcnew Label();
   l->Text = "Hallo Welt";
   l->AutoSize = true;
   l->Left = 35;
   l->Top = 105;
   l->Font = gcnew System::Drawing::Font("Arial", 33);
   l->TextAlign = ContentAlignment::MiddleCenter;
   l->Anchor = AnchorStyles::None;
  f->ShowDialog();
}
```

Mit der ersten Anweisung setzen Sie die Beschriftung in die Mitte des Label-Steuerelements und mit der zweiten stellen Sie sicher, dass die Position des Beschriftungsfeldes relativ zu den Rändern

des Dialogfensters gleich bleibt, wenn dieses seine Größe – durch Maximieren oder durch Ziehen mit der Maus – ändert.

`ContentAlignment` ist eine so genannte Enumeration – das Wort bedeutet zu Deutsch *Aufzählung* und genau das ist es auch. Eine Aufzählung stellt verschiedene Konstanten zur Verfügung, die Sie im Code benutzen können. Die Konstante der `ContentAlignment`-Enumeration, die wir in diesem Beispiel benötigen, ist `MiddleCenter`. Der Zugriff erfolgt in bekannter Weise über den Scope-Operator.

Genauso verhält es sich mit `AnchorStyles`. Diese Enumeration enthält Konstanten, mit denen Sie die Art der Verankerung eines Steuerelements an den Rändern des übergeordneten Containers festlegen. Die für uns passende Konstante ist hier `None`, da das Beschriftungsfeld an keinen Fensterrand gebunden sein soll – nur dann behält es im Falle einer Größenänderung seine Position in der Mitte.

Zu guter Letzt fügen Sie das Label im Code dem Dialogfenster hinzu. Dazu verwenden Sie die Methode `Add()` der `Controls`-Eigenschaft des Formular-Objekts. Diese Eigenschaft repräsentiert alle auf einem Formular angeordneten Steuerelemente.

8. Fügen Sie der *button1_Click()*-Methode die Anweisung `f->Controls->Add(l);` hinzu. Damit steht Ihr Programm.

```
System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
   Form ^f = gcnew Form();
   f->Width = 300;
   f->Height = 300;
   Label ^l = gcnew Label();
   l->Text = "Hallo Welt";
   l->AutoSize = true;
   l->Left = 35;
   l->Top = 105;
   l->Font = gcnew System::Drawing::Font("Arial", 33);
   l->TextAlign = ContentAlignment::MiddleCenter;
   l->Anchor = AnchorStyles::None;
   f->Controls->Add(l);
  f->ShowDialog();
}
```

9. Starten Sie das Programm in der Visual C++ 2010 Express-Oberfläche mit F5 oder Strg+F5.
10. Klicken Sie auf die Schaltfläche mit der Aufschrift *KLICK MICH*. Jetzt sollte sich das Dialogfenster in ähnlichem Design wie im Beispiel aus dem letzten Kapitel präsentieren.

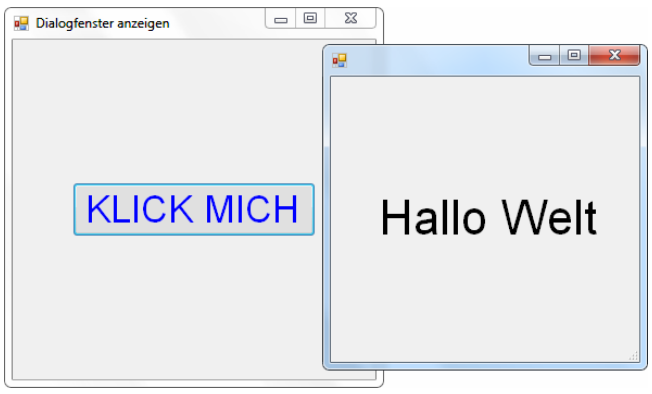

Abbildung 5.10: Über Programmcode konfiguriertes Dialogfenster

Weitere Steuerelemente

Visual C++ 2010 Express stellt Ihnen als Entwickler von Windows-Anwendungen eine Vielzahl von Steuerelementen zur Verfügung, wie Sie sie als Anwender im Umgang mit professionellen Programmen gewöhnt sind. Daneben können Sie später im Code sogar eigene Steuerelemente definieren. Das ist gar nicht so schwer, wie es sich vielleicht anhört. Sie leiten das Steuerelement von einer bestehenden Steuerelement-Klasse ab und fügen ihm dann zusätzliche Funktionalität in Form von Methoden und Eigenschaften hinzu. (Wie man eine Klasse von einer anderen ableitet, werden Sie in den Kapiteln über die objektorientierte Programmierung erfahren.)

Allerdings wird sich nur in ganz speziellen Fällen die Notwendigkeit ergeben, eigene Steuerelemente zu entwickeln und es ist sogar so, dass man von vielen der Steuerelemente, die Visual C++ Express bereitstellt, im Allgemeinen gar keinen Gebrauch macht. Andere Steuerelemente wiederum gehören praktisch zum ständigen Repertoire eines .NET-Entwicklers. Tabelle 5.2 zeigt eine Auflistung der Steuerelemente, die Sie vermutlich am häufigsten verwenden werden.

Tabelle 5.2: Gebräuchliche Steuerelemente mit Beschreibung

Steuerelement	Beschreibung
Label	Beschreibungstext, in der Regel als Beschriftung für andere Steuerelemente gedacht
TextBox	Textfeld zur Eingabe von Text
RichTextBox	Textfeld mit umfangreichen Formatierungsmöglichkeiten
Button	Schaltfläche zum Anklicken
RadioButton	Optionsschaltfläche, die aktiviert oder deaktiviert werden kann
CheckBox	Kontrollkästchen, das aktiviert oder deaktiviert werden kann
MenuStrip	Zeigt eine obere Menüleiste in bekannter Form, auch kaskadiert, an
ToolTip	Quickinfo für andere Steuerelemente

RadioButton- und CheckBox-Steuerelemente besitzen eine ähnliche Funktion. Beide stellen Auswahlmöglichkeiten bereit, die ein Benutzer durch Anklicken aktivieren oder deaktivieren kann. Während sich die Optionsfelder eines RadioButton-Steuerelements jedoch gegenseitig ausschließen – d.h. von mehreren vorhandenen Optionen kann vom Benutzer jeweils nur eine ausgewählt werden –, können gleichzeitig mehrere CheckBoxen (Kontrollkästchen) aktiviert sein. Mehrere RadioButton-Steuerelemente eines Formulars oder in einem so genannten Panel (einem weiteren Steuerelement) zusammengefasste RadioButton-Steuerelemente bilden dabei eine Gruppe. Wenn ein Benutzer ein Optionsfeld innerhalb einer Gruppe aktiviert, werden die anderen Optionsfelder automatisch deaktiviert.

MenuStrip-Steuerlemente erzeugen am oberen Rand einer Form ein Standardmenü. Zum Hinzufügen eines MenuStrip-Steuerelements in Ihr Formular gehen Sie wie gewohnt vor: Ziehen Sie das Steuerelement einfach auf das Formular oder klicken Sie je einmal in der Toolbox auf das MenuStrip-Symbol bzw. auf den Text *MenuStrip* und danach an eine beliebige Position innerhalb des Formulars. Danach erscheint im unteren Bereich des Entwurfsfensters – dem so genannten Komponentenfach – ein Eintrag für das MenuStrip-Steuerelement.

Das Komponentenfach dient zur Aufbewahrung von Steuerelementen, die zur Laufzeit entweder nicht sichtbar sind, was zum Beispiel auf einen Timer zutrifft, oder die einen festen Platz haben, wie es bei einem Standardmenü der Fall ist. Das Komponentenfach ist entsprechend nur im Entwurfsmodus, nicht aber zur Laufzeit sichtbar.

Nach dem Ablegen des MenuStrip-Steuerelements auf dem Formular wird Ihnen oben in der Menüleiste für die erste Auswahlmöglichkeit ein Textfeld angeboten. (Klicken Sie gegebenenfalls das MenuStrip-Steuerelement im Komponentenfach an, falls diese Anzeige inzwischen nicht mehr sichtbar ist.) Sie können sogleich mit der Eingabe des Textes für die erste Auswahlmöglichkeit beginnen. Sie müssen dazu nicht extra in die Menüleiste klicken; es genügt, wenn das MenuStrip-Steuerelement aktiv ist. Mit dem ersten Tastendruck wird die Einfügemarke in ein dann sichtbares Textfeld der Menüleiste gesetzt. Beenden Sie die Eingabe mit *Enter*. Danach erscheint rechts davon ein weiteres Textfeld für die nächste Auswahlmöglichkeit und unterhalb davon ein Textfeld für Auswahlmöglichkeiten der zweiten Ebene, vergleichbar zum Beispiel mit den Menüauswahlen *Datei* oder *Bearbeiten* für die erste Ebene bzw. *Bearbeiten/Kopieren* für die zweite Ebene, wie sie in fast jedem Windows-Programm vorhanden sind. Durch Drücken der Eingabetaste werden Ihnen jeweils neue leere Felder zum Erstellen weiterer Auswahlmöglichkeiten angeboten. Sobald Sie das Menü vollständig aufgefüllt haben, entziehen Sie dem MenuStrip-Steuerelement den Fokus, indem Sie mit der Maus einfach in einen anderen Bereich des Formulars klicken.

Um ein MenuStrip-Steuerelement nachträglich weiter zu bearbeiten, klicken Sie das Steuerelement im Komponentenfach an oder Sie klicken in die obere Menüleiste. Um einem bestimmten Menüeintrag weitere Untereinträge hinzuzufügen, klicken Sie auf den Eintrag der übergeordneten Ebene. Mit der Taste Entf lassen sich Menüeinträge nachträglich entfernen.

ToolTip-Steuerelemente können Sie fast jedem beliebigen anderen Steuerelement zuordnen. Sie zeigen eine Quickinfo an, wenn ein Benutzer den Mauszeiger auf das betreffende Steuerelement bewegt und ein paar Augenblicke wartet.

Hinweis: »ToolTip auf ...«-Eigenschaft

Wenn Sie ein ToolTip-Steuerelement in Ihrer Form ablegen, wird allen Steuerelementen – auch dem Formular selbst – eine Eigenschaft `ToolTip auf <Name_des_Tooltip_Elements>` hinzugefügt. Um für ein bestimmtes Steuerelement eine Quickinfo einzurichten, müssen Sie dieser Eigenschaft im Eigenschaftenfenster lediglich einen Wert zuweisen. Sie benötigen also keinesfalls für jedes Steuerelement ein eigenes ToolTip-Steuerelement.

Eine Quickinfo für den »KLICK MICH«-Button richten Sie demnach wie folgt ein:

1. Ziehen Sie ein ToolTip-Steuerelement auf das Formular.

   Das ToolTip-Steuerelement finden Sie in der Toolbox unter *Allgemeine Steuerelemente* oder unter *Alle Windows Forms*.
2. Selektieren Sie das Button-Steuerelement auf dem Formular.
3. Tippen Sie im Eigenschaftenfenster für die Eigenschaft `ToolTip auf toolTip1` des Button-Steuerelements den Text **Zeigt ein Dialogfenster an** ein.
4. Selektieren Sie im Formular das ToolTip-Steuerelement.
5. Legen Sie als Wert für die Eigenschaft `ToolTipTitle` dieses Steuerelements den Text **Info** fest.
6. Setzen Sie zusätzlich den Wert von `ToolTipIcon` auf *Info*.

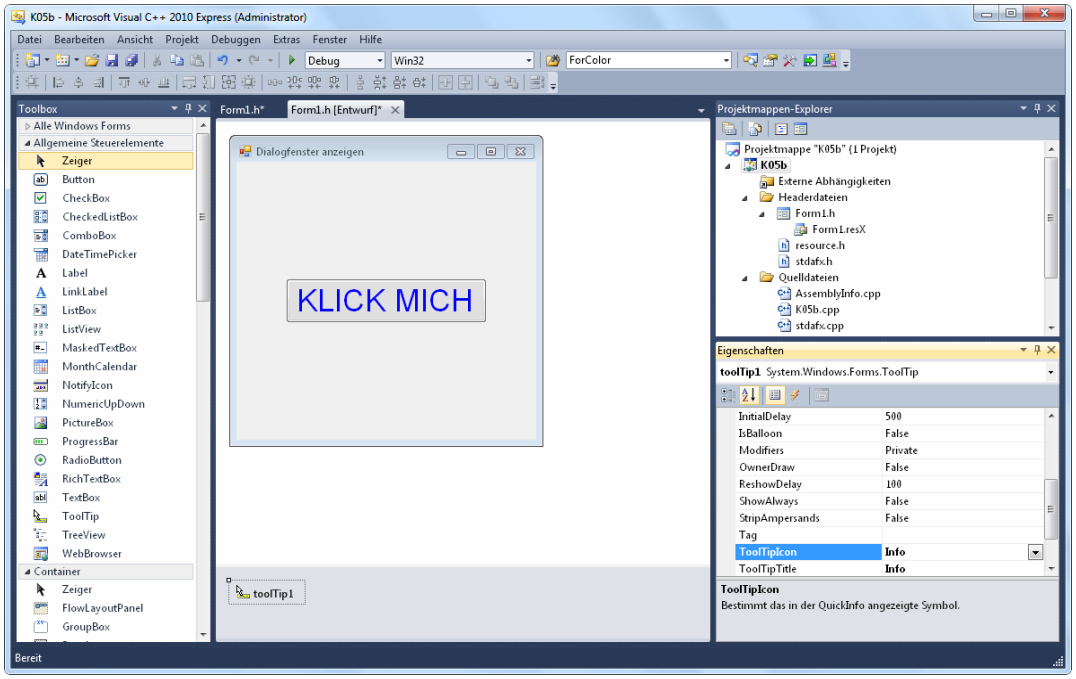

Abbildung 5.11: Komponentenfach mit ToolTip-Steuerelement

7. Starten Sie das Programm erneut.
8. Bewegen Sie den Mauszeiger auf die Schaltfläche und warten Sie einen Moment.

   Nun sollte sich eine Quickinfo zeigen, wie sie in Abbildung 5.12 zu sehen ist.

Abbildung 5.12: Button-Steuerelement mit Quickinfo

5.4 MessageBoxen

Da wir gerade bei Dialogfenstern sind, möchte ich Ihnen ein ganz spezielles, überaus nützliches Exemplar vorstellen: die so genannte MessageBox. Dabei handelt es sich um ein Meldungsfenster, das Sie über den Aufruf einer simplen Methode erzeugen und konfigurieren können (dazu gleich mehr).

Die MessageBox wird sehr häufig verwendet und kann Ihnen zum Beispiel auch bei der oben genannten Kontrolle – sich schnell anzeigen zu lassen, wann genau ein bestimmtes Ereignis eintritt – behilflich sein.

Zur Demonstration verwenden wir das aktuelle Beispielprojekt.

1. Selektieren Sie das Anwendungsformular in der Entwurfsansicht.
2. Wechseln Sie im Eigenschaftenfenster in die Ereignisansicht.
3. Klicken Sie doppelt auf das `DoubleClick`-Ereignis.

   Visual C++ 2010 Express stellt Ihnen das Grundgerüst der korrespondierenden Ereignismethode zur Verfügung.

   ```
   System::Void Form1_DoubleClick(System::Object^  sender, System::EventArgs^  e) {
   }
   ```

   Eine MessageBox erzeugen Sie mit der Methode `Show()` der Klasse `MessageBox` – daher der Name. Das Meldungsfenster erscheint standardmäßig ohne Titel und mit einer OK-Schaltfläche. Beides – den Text der Titelzeile sowie Anzahl und Art der Schaltflächen – können Sie nach Bedarf ändern, indem Sie `Show()` beim Aufruf entsprechende Argumente[6] übergeben.

   Mit dem ersten Argument legen Sie den Meldungstext und mit dem zweiten gegebenenfalls den Text für die Titelzeile fest. Der dritte Parameter ist wie der zweite optional. Mit ihm können Sie Anzahl und Art der angezeigten Schaltflächen bestimmen. Dazu stehen Ihnen die Konstanten `OK` (das ist der Standard) `OKCancel` (Schaltflächen *OK* und *Abbrechen*), `YesNo` und `AbortRetryIgnore` (Abbrechen, Wiederholen, Ignorieren) zur Verfügung.

   Eine genaue Beschreibung der Methode finden Sie in der MSDN-Hilfe unter *http://msdn.microsoft.com/de-de/library/system.windows.messagebox.show(v=VS.100).aspx* (Softlink **cpp0501**). Alternativ verwenden Sie in der MSDN Suche die entsprechenden Suchbegriffe (z.B. »MessageBox::Show« bzw. »MessageBox.Show«; in der MSDN-Hilfe kommt das auf das Gleiche hinaus, da in den anderen .NET Sprachen Methoden ausschließlich mit dem Punktoperator referenziert werden).
4. Fügen Sie die Anweisung `MessageBox::Show("Sie haben doppelt auf das Formular geklickt", "Ereignis DoubleClick");` in der Ereignismethode *Form1_DoubleClick()* ein.

   ```
   System::Void Form1_DoubleClick(System::Object^  sender, System::EventArgs^  e) {
      MessageBox::Show("Sie haben doppelt auf das Formular geklickt", "Ereignis DoubleClick");
   }
   ```
5. Starten Sie Ihr Programm.
6. Klicken Sie im Formular doppelt auf eine freie Stelle.

   Daraufhin sollte ein Meldungsfenster wie in Abbildung 5.13 zu sehen erscheinen.

[6] Argumente nennt man die Werte, die einer Methode beim Aufruf übergeben werden. Eine andere Bezeichnung ist Parameter. Genau genommen ist diese Bezeichnung jedoch mehrdeutig, wie Sie in Kapitel 8 noch erfahren werden. Der Begriff Argument ist etwas exakter.

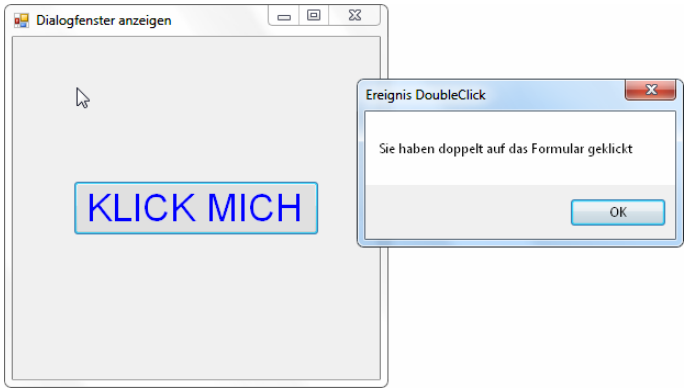

Abbildung 5.13: Mithilfe der Methode `Show()` der Klasse `MessageBox` können Sie auf einfache Weise Meldungsfenster zur Anzeige bringen

Wie Sie sich denken können, ist die Methode `Show()` der Klasse `MessageBox` – diese ist im Namensraum `System::Windows::Forms` definiert – gewissermaßen das Pendant der Windows Forms zu den Ausgabemethoden `Write()` und `WriteLine()` der `Console`-Klasse, die Sie in Kapitel 3 kennen gelernt haben.

5.5 Übungen zu diesem Kapitel

In diesem Abschnitt finden Sie wieder Übungen zu diesem Kapitel. Die Lösungen finden Sie im Beispielordner *Beispiele/K05/Lösungen* sowie auf der Webseite *http://www.richtig-einsteigen.de*.

Übung 5.1

Ergänzen Sie das Beispiel *HalloWelt_GUI* aus Abschnitt 5.1 so, dass sich die Schrift auf einen Mausklick hin rot einfärbt. Dazu müssen Sie das Ereignis `Click` des `Label`-Steuerelements entsprechend bearbeiten[7].

Übung 5.2

Implementieren Sie im gleichen Beispiel für das Ereignis `DoubleClick` des `Label`-Steuerelements ein Meldungsfenster. Der Meldungstext für dieses Ereignis soll lauten: »Sie haben mich doppelt angeklickt«.

[7] Zwischen den Ereignissen `Click` und `MouseClick` besteht so gut wie kein praktischer Unterschied. Genauso verhält es sich zwischen den Ereignissen `DoubleClick` und `MouseDoubleClick`.

5.6 Zusammenfassung

Sie haben in den letzten beiden Kapiteln einiges über Ihre Entwicklungsumgebung erfahren und die wichtigsten Funktionen in der Visual C++ 2010 Express-Oberfläche dürften Ihnen nunmehr vertraut sein. Sie haben auch schon etwas über Ereignisprogrammierung gehört und wissen, wie Sie in Ihren Programmen zusätzliche Dialogfenster integrieren. Dabei haben Sie schon verschiedene fortgeschrittene Features verwendet, ja, Sie haben sogar im Programmcode Objekte von Klassen instanziert.

Es lässt sich jetzt leider nicht vermeiden, dass wir Sie in den nächsten Kapiteln mit einer Menge Theorie, vor allem zur Programmiersprache C++, konfrontieren. Aber wir werden zwischendurch das Theoretische mit praktischen Beispielen unterlegen.

Verlieren Sie also nicht den Spaß am Programmieren. Ich versichere Ihnen, dass dieser mit fortgeschrittenen Kenntnissen noch steigen wird.