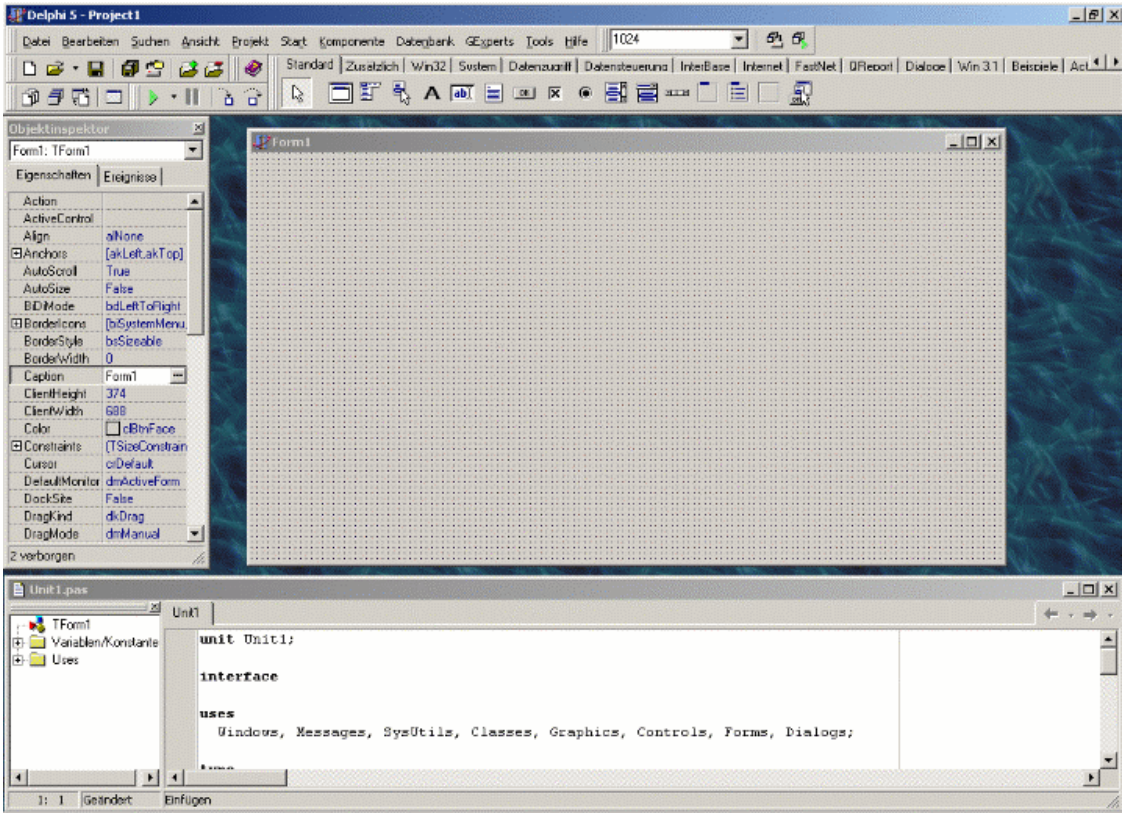
Delphi-Source.de - Grundlagen

Delphi-IDE

Einführung

Startet man Delphi, erhält man - je nach Anordnung der einzelnen Fenster und abgesehen vom Desktop-Hintergrund - folgenden Aufbau, die IDE (Integrated Development Environment, Integrierte Entwicklungsumgebung):



Den Einsteiger mag das etwas verwirren, besonders, wenn er sich durch die langen Menüs bewegt. Wir wollen hier die Möglichkeiten von Delphi ein wenig näher beschreiben. Dabei soll jedoch nicht auf jedes Detail eingegangen werden. Stattdessen sollen die wichtigsten Funktionen, die auch von Einsteigern benötigt werden, beschrieben werden. Hat man sich erst einmal in die Bedienung der IDE eingearbeitet, sollte es nicht mehr allzu schwer sein, auch die weiteren Funktionen zu verwenden.

Die Beschreibung und die Screenshots orientieren sich an Delphi 5 Professional. Der Aufbau von Delphi hat sich in den älteren Versionen nicht grundlegend geändert, weshalb das Meiste auch problemlos mit älteren Versionen nachvollzogen werden kann. Es ist allerdings durchaus möglich, dass Dialogfenster anders aussehen, Menüpunkte in anderen Menüs zu finden sind oder evtl. auch noch gar nicht existieren. Dies sollte allerdings die Ausnahme sein.

Um noch einmal auf den Begriff "IDE" zurückzukommen: Das "Integrierte" daran ist der Zugriff auf Editor, Compiler, Linker und Debugger unter einer einzigen Oberfläche. Im Gegensatz dazu stehen die reinen Compiler, wobei der Programmcode in einem normalen Texteditor geschrieben wird und anschließend dem Compiler auf Befehlszeilenebene als Parameter übergeben wird.

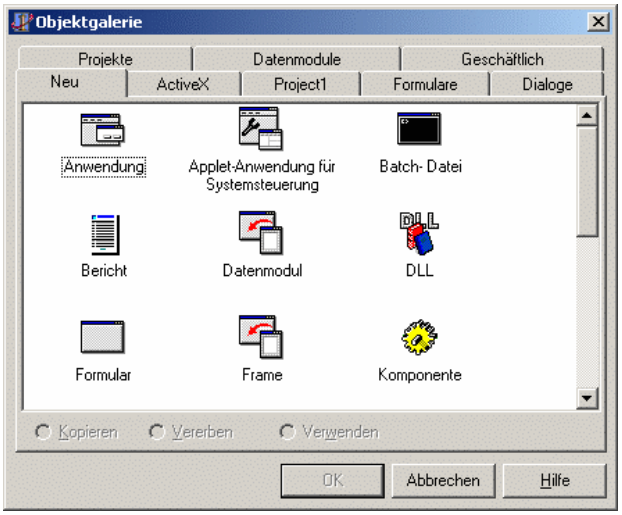
Das Konzept der integrierten Entwicklungsumgebung ist also sehr komfortabel und vor allem Zeit sparend.

Projektverwaltung

Delphi-Projekte

Anfänger in Delphi haben häufig Probleme mit den vielen Dateien, die Delphi automatisch erzeugt und die später gespeichert werden müssen. Um dies zu klären, gehen wir einen solchen Fall Schritt für Schritt durch:


Über dieses Symbol (alternativ Menü Datei/Neu...) erhalten wir das folgende Dialogfenster, genannt "Objektgalerie":



Da wir eine neue Anwendung erstellen wollen, wählen wir hier diesen Punkt auch aus, es ist gleich der erste. Nach Bestätigung mit OK erhalten wir das in der Einführung gezeigte Bild.

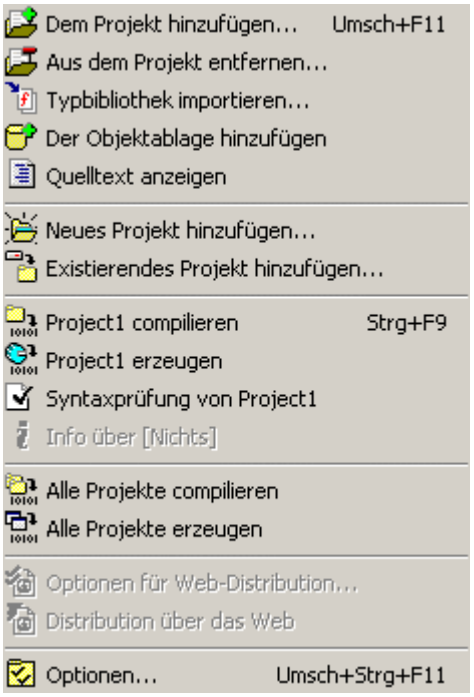

Klicken wir nun auf das Symbol mit den mehreren Disketten, um wirklich alle Dateien zu speichern, werden wir nacheinander nach zwei Dateinamen gefragt. Der erste bekommt die Endung pas und bezeichnet den Quellcode, den wir in der unteren Bildschirmhälfte gesehen haben. Der zweite ist vom Typ "Delphi-Projekt" und bekommt die Endung dpr. Wenn wir unsere Anwendung später kompilieren, erhält sie den gleichen Namen - allerdings mit der Endung exe.
Haben wir diese Dateien in einem neuen Verzeichnis gespeichert, was für jedes Projekt sinnvoll ist, finden wir dort nun aber nicht nur zwei, sondern gleich mehrere Dateien.

DPR	**Delphi Project** Ein Projekt (also eine Anwendung) besteht in der Regel aus mehreren Quellcode-Dateien und Fenstern. In der Projektdatei wird gespeichert, was alles zum Kompilieren benötigt wird. Außerdem enthält die Datei den Quellcode des Hauptprogramms, das (in der Regel) das erste Fenster auf den Bildschirm bringt. Es ist normalerweise nicht nötig, diese Datei von Hand zu bearbeiten.
PAS	**Pascal-Quellcode** Diese Datei kann mit jedem Texteditor geöffnet werden und enthält eine Pascal-Unit.

DFM	**Delphi Form** Jedes Fenster einer Anwendung (Form bzw. Formular genannt) hat seine eigene Unit mit dem zugehörigen Quellcode. Beim Speichern vergibt Delphi der Fenster-Datei automatisch den gleichen Namen wie dem Quellcode, nur mit der Endung dfm. In ihr sind alle Eigenschaften eines Formulars abgelegt, und sie sollte nicht von Hand, sondern nur über die grafische Oberfläche bzw. den Objektinspektor bearbeitet werden.
DOF / CFG	**Konfigurationsdatei (Delphi Option File, Configuration)** Diese beiden Dateien enthalten Einstellungen zu einem Projekt, wie Verzeichnisse und Informationen, um den zuletzt angezeigten Zustand beim Neuladen in Delphi wiederherzustellen.

Um ein Delphi-Projekt weiter zu bearbeiten, reicht es aus, die Projektdatei zu laden (Datei/Projekt öffnen...).

Das Projekt-Menü



Hat man ein neues Projekt angelegt, lässt dieses sich über das Projekt-Menü bearbeiten.

Zu allererst lassen sich über den Menüpunkt "**Dem Projekt hinzufügen**" bereits vorhandene Units eingliedern. Ebenso lassen sich über "**Aus dem Projekt entfernen**" Units wieder ausgliedern. Sie werden dabei nicht physikalisch gelöscht, sondern sind nur nicht mehr in der Projektdatei enthalten.

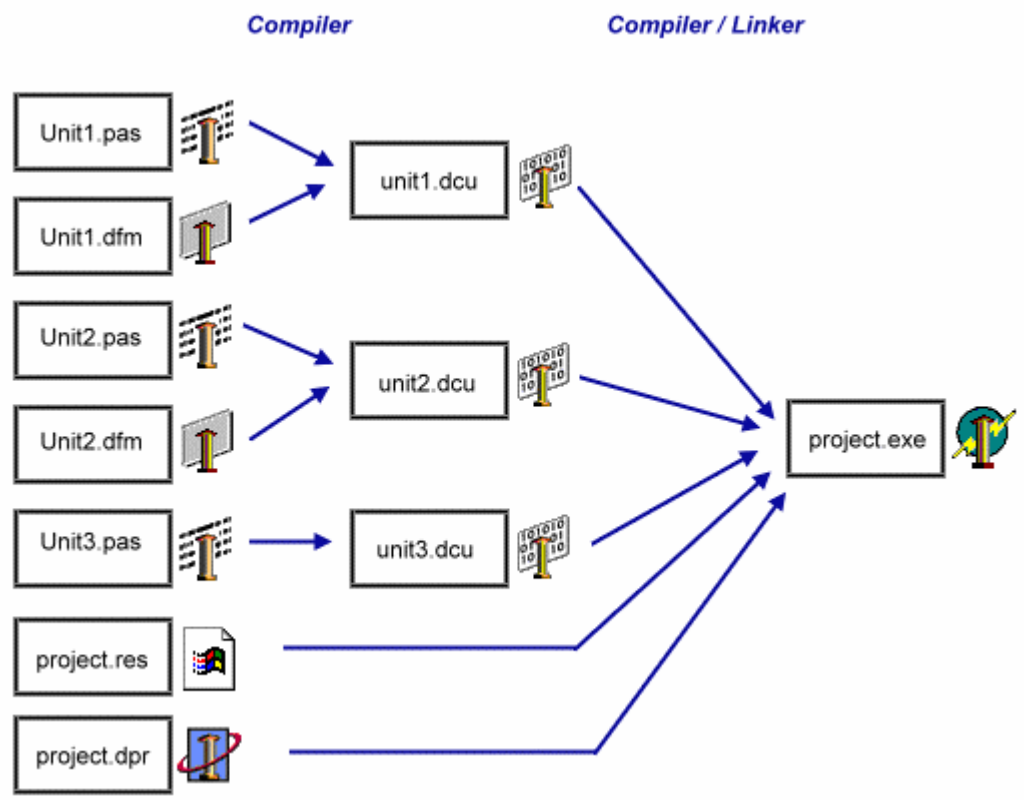
Hat man einmal ein so tolles Fenster entworfen, dass man es in anderen Projekten auch verwenden will, lässt es sich über den Menüpunkt "**Der Objektablage hinzufügen**" so speichern, dass es in dem "Datei/Neu..." Fenster (Objektgalerie) auftaucht und als Vorlage für ein neues Fenster verwendet werden kann.

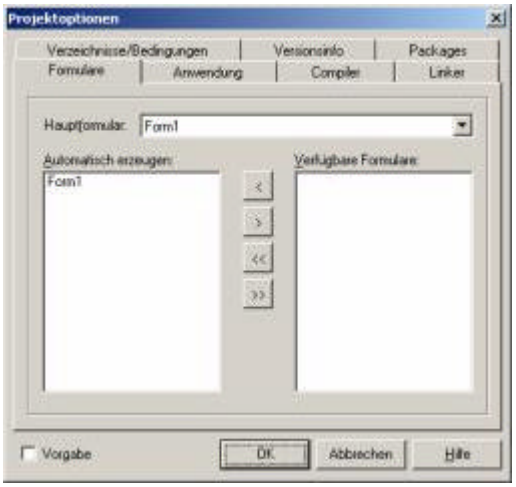
Über "**Quelltext anzeigen**" gelangt man schließlich zum Quellcode der geheimnisvollen Projektdatei. Manchmal kann es durchaus sinnvoll sein, hier etwas einzufügen (z. B. wenn etwas ausgeführt werden soll, bevor das erste Fenster angezeigt wird). Generell ist dies aber mit großer Vorsicht zu tun, da die Datei von Delphi automatisch verwaltet wird und es dabei dann evtl. zu Problemen kommen kann.

Die folgenden Menüpunkte sind interessant, wenn man ein Produkt erstellen will, das aus mehreren Delphi-Projekten besteht (z. B. einer EXE und mehreren DLLs). In älteren Delphi-Versionen ließ sich keine Verbindung zwischen einzelnen Projekten herstellen. Auch jetzt ist es nicht möglich, mehr als ein Projekt gleichzeitig geöffnet zu haben. Allerdings lassen sich Projektgruppen (.bpg) erstellen, die mehrere Projekte enthalten.

Nun kommen wir zu den wichtigsten Funktionen überhaupt: **Projekt compilieren** und **Projekt erzeugen**. Hierdurch (oder einfach auch durch Strg+F9) wird aus allen Dateien eines Projekts eine ausführbare Anwendungen (.exe) erzeugt. Der Unterschied zwischen den beiden Funktionen liegt darin, dass "Projekt compilieren" immer nur zwischenzeitlich veränderte Units/Formulare neu kompiliert, während bei "Projekt erzeugen" alles komplett neu kompiliert wird. Bei der ebenfalls angebotenen "**Syntaxprüfung**" wird nur auf Compilerfehler geprüft und keine EXE erzeugt, was deshalb (besonders bei großen Projekten) wesentlich schneller geht.
Durch das Kompilieren kommen weitere Dateiendungen ins Spiel:

DCU	**Delphi Compiled Unit** Eine kompilierte Unit entsteht durch das Kompilieren einer pas und evtl. der zugehörigen dfm-Datei.
EXE	**Executable** Nun kommt neben dem Compiler auch der Linker ins Spiel, der aus allen DCUs und den RES-Dateien eine ausführbare EXE im Maschinencode (bzw. eine DLL) erstellt.





Das Wichtigste neben dem Kompilieren ist natürlich das **Starten der Anwendung** aus Delphi heraus. Dies ist über das Menü Start/Start möglich. Man wird sich aber schnell daran gewöhnt haben, nur noch die Taste F9 zu drücken. Sollen dem Programm **Startparameter** übergeben werden, so lassen diese sich unter Start/Parameter vorgeben. Das Eingabefeld "Host" wird nur bei DLLs benötigt. Hier wird dann der Name der ausführbaren Datei eingegeben, die die DLL benötigt.

Interessant ist auch noch der letzte Menüpunkt "**Optionen**" (s. Fenster links):

Gleich auf der ersten Register-Seite (**Formulare**) lassen sich einzelne Fenster von der automatischen Erzeugung auszuschließen, um den Start der Anwendung zu beschleunigen.

Solche Fenster müssen später über

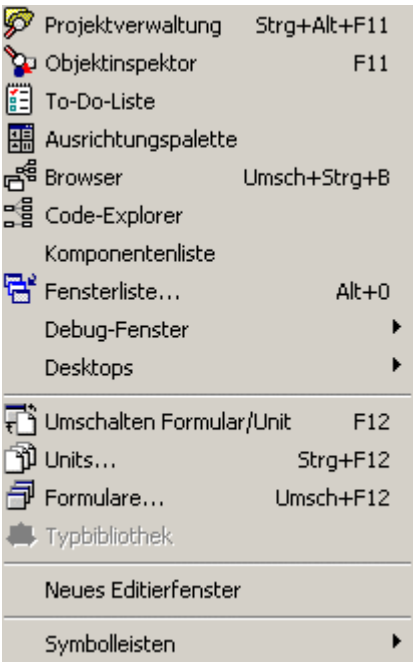
```
try
  Form2:=TForm2.Create(Application);
  Form2.ShowModal;
finally
  Form2.Release;
end;
```

erzeugt werden.

Unter "**Anwendung**" kann dem Programm ein beschreibender Namen (nicht Name der EXE, wird im Explorer im Datei-Eigenschaften-Dialog angezeigt) sowie ein Icon und eine Hilfe-Datei zuordnen. Außerdem lassen sich hier Compiler-, Linker- und Verzeichniseinstellungen vornehmen.

Unter **Versionsinfo** kann zusätzlich eine Versionsnummer in die EXE übernommen werden, die dann vom Benutzer im Datei-Eigenschaften-Dialog oder auch programmseitig (s. Tipps & Tricks) überprüft werden kann.

Menü Ansicht



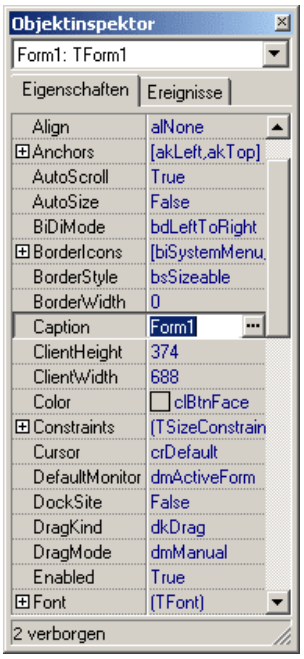
Nicht vergessen werden darf auch das Menü Ansicht.

Hier findet man gleich unter dem ersten Menüpunkt eine Anzeige aller im Projekt vorhandenen Units und Formulare samt Speicherort.

Die weiteren hier wichtigen Menüpunkte in der unteren Hälfte des Menüs sind auch direkt über die Symbolleiste zugänglich wie das **Umschalten zwischen Formular und zugehöriger Unit** (noch schneller ist hier die Taste F12) sowie die **Anzeige aller Units** (zum Wechseln in eine andere Quellcodedatei) und Formulare.

Projekte erweitern

Um weitere Fenster oder auch reine Units in ein Projekt einzufügen ist wieder das Datei-Menü von Nutzen. Über den Menüpunkt "Neu..." lassen sich hier weitere Elemente in ein Projekt übernehmen, wie z. B. auch Datenmodule als Container für nicht visuelle Datenbankkomponenten (TTable, TDataSource usw.).



VCL/Formdesigner/Objektinspektor

Kommen wir nun zum Kernstück der visuellen Programmierung.

Beim Anlegen eines neuen Projekts erzeugt Delphi automatisch eine erste Unit mit Formular. Dieses Formular wird das Hauptfenster unserer Anwendung.
Zwischen Quellcode und Fenster kann mit der Taste F12 hin- und hergewechselt werden.

Bevor das Fenster mit Inhalt gefüllt wird, sollte es erst selbst konfiguriert werden. Um Eigenschaften visueller Komponenten zur Entwurfszeit zu verändern, muss keine Zeile Code getippt werden. Dafür wird der **Objektinspektor** verwendet. Befinden sich auf dem Formular bereits irgendwelche Elemente, so werden sie mit einfachem Mausklick markiert, und schon zeigt der Objektinspektor die Eigenschaften dieses Elements an. Klickt man auf eine leere Fläche des Fensters, betreffen die Eigenschaften das Fenster selbst.

Eigenschaften im Objektinspektor

Da unser Fenster noch leer ist, wird im Objektinspektor automatisch Form1 angezeigt.
"Form1" ist der Name dieses Fensters, er lässt sich im Objektinspektor unter "Name" verändern, wobei die üblichen Regeln für Variablennamen gelten (keine Umlaute, Leerzeichen usw.). Dieser Name ist nur für den Programmierer interessant. Über ihn kann das Fenster aus dem Programmcode heraus angesprochen werden. Für den Benutzer des Programms ist dieser Name nicht sichtbar.

Der Benutzer sieht etwas anderes: Die Überschrift des Fensters in der blauen Titelzeile. Diese wird im Objektinspektor unter "Caption" verändert. Tippt man hier einen beliebigen Text ein, ändert sich augenblicklich auch die Titelzeile im angezeigten Fenster.

Des Weiteren ist es sinnvoll, die Eigenschaft BorderStyle auf bsSingle zu setzen, wenn man nicht will, dass der Benutzer das Fenster in der Größe verändern kann.
Außerdem kann festgelegt werden, dass das Fenster im Vollbildmodus angezeigt wird (WindowState:=wsMaximized) oder zentriert auf dem Desktop erscheinen soll (Position:=poDesktopCenter).

Sehen Sie sich aber auch die weiteren Eigenschaften von TForm (Fensterklasse) an, Hilfe erhalten Sie, indem Sie zuerst eine Eigenschaft markieren und anschließend F1 drücken.

Das Fenster füllen

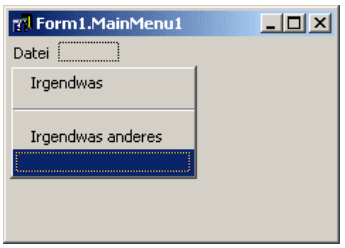
Die Möglichkeit, visuelle Komponenten zu verwenden, um eine ansprechende Benutzeroberfläche zu erstellen, erlaubt es Ihnen, sich mehr auf den eigentlichen Programmcode zu konzentrieren und nicht einen großteil der Zeit mit der Erstellung einer Oberfläche zu "verschwenden".

In dem oberen Fenster der Delphi-Entwicklungsumgebung finden Sie die Komponentenpalette, die standardmäßig schon zig Komponenten enthält. Es handelt sich hierbei um die **VCL**, die "**Visual Component Library**".
Unter den vielen Komponenten, die sich hier befinden, gibt es sichtbare und nicht sichtbare. Die sichtbaren (z. B. TEdit auf der Seite "Standard") sehen Sie bei der Entwicklung genau so, wie sie auch später der Benutzer sieht. Andere dagegen (z. B. TTimer auf der Seite "System") sind vom Benutzer nicht zu sehen. Nur der Entwickler sieht ein kleines quadratisches Symbol auf seinem Fenster, für das er die Eigenschaften im Objektinspektor festlegen kann.

Für fast jeden Wunsch gibt es eine passende Komponente. Wenn sie nicht bei Delphi dabei ist, so kann man sie in einer großen Auswahl im Internet finden. Links zu den einschlägigen Website finden Sie in unserer Link-Sammlung. Wie diese Komponenten installiert werden, erfahren Sie in einem späteren Abschnitt.

Zunächst setzen Sie willkürlich ein paar Standard-Komponenten auf Ihr noch leeres Fenster. Dazu klicken Sie eine Komponente in der Symbolleiste an und klicken anschließend auf das Fenster. An dieser Stelle erscheint sie nun in der Standardgröße. Mit der Maus kann sie vergrößert oder verkleinert werden - oder auch über die Eigenschaften height und width im Objektinspektor.

Sonderfall: MainMenu

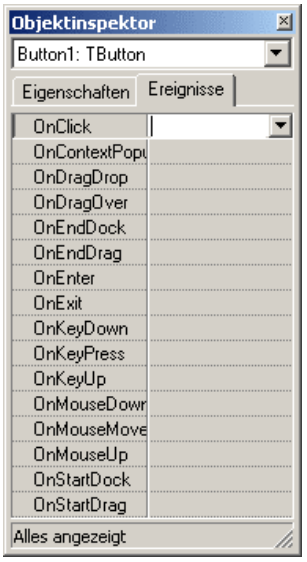


Einige spezielle Komponenten enthalten einen eigenen Editor. Setzen Sie z. B. die Komponente "MainMenu" auf das Fenster und klicken mit der rechten Maustaste darauf. Hier sehen Sie nun als ersten Eintrag den "Menü-Designer". Öffnen Sie ihn und geben gleich dem ersten Menüpunkt über die Eigenschaft "Caption" einen Namen. Nach Drücken der Return-Taste erscheint im Designer automatisch ein leerer Eintrag darunter. Dies wird immer so sein, damit einfach weitere Einträge angehängt werden können. Der leere Eintrag am Ende ist zur Laufzeit natürlich nicht zu sehen. Die Menüpunkte können auch mit der Maus verschoben werden. Geben Sie bei einem Menüpunkt als Caption das Minuszeichen "-" ein, erscheint ein durchgezogener, horizontaler Trennstrich.

Tabulatorreihenfolge

Sicher ist Ihnen bekannt, dass man in Dialogfenstern auch mit Hilfe der Tabulator-Taste von einem Feld ins nächste wechseln kann. Das ist standardmäßig auch für Delphi-Anwendungen so. Nur kann es passieren, dass im späteren Verlauf noch ein Button oder ein Edit-Feld zwischendrin eingefügt werden muss. Dieses wird vom Tabulator übersprungen und erhält den Cursor als letzte Komponente. Das lässt sich ändern, indem Sie mit der rechten Maustaste auf eine freie Stelle des Formulars klicken und "Tabulatorreihenfolge" auswählen. Hier können Sie nun frei verschieben, welche Komponenten der Reihe nach angesprungen werden sollen.

Ereignisse im Objektinspektor



Sicher haben Sie bemerkt, dass es im Objektinspektor nicht nur die Seite "Eigenschaften" gibt, sondern auch noch "Ereignisse". Hier legen Sie fest, wie eine Komponente bei bestimmten Aktionen des Benutzers reagieren soll.
Gutes Beispiel hierfür ist ein Button, der beim Anklicken eine Aktion ausführen soll.
Setzen Sie also einen Button auf das Formular und wechseln im Objektinspektor auf "Ereignisse". Hier sind nun alle Ereignisse aufgeführt, auf die dieser Button reagieren könnte. Eine Erklärung zu den einzelnen bekommen Sie dadurch, dass Sie einen Eintrag markieren und anschließend F1 drücken.

Wir wollen, dass auf einen Mausklick reagiert wird - hierbei handelt es sich um das Ereignis OnClick. Klicken wir also doppelt auf den freien Bereich rechts neben dem Eintrag "OnClick". Dadurch wird automatisch im Quellcode der zugehörigen Unit eine Methode angelegt. Zwischen den Wörtern begin und end steht noch nichts, dafür sind wir zuständig.
Nehmen wir also an, der Button soll inaktiv werden, wenn wir einmal darauf geklickt haben. Die Eigenschaft für aktiv/inaktiv einer Komponente lautet "enabled". Wir ergänzen die Methode also wie folgt:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Enabled:=false;
end;
```

Auf diese Art in den Code eingefügte Ereignisbehandlungsmethoden sollten nicht von Hand gelöscht werden, da es dadurch Probleme geben kann. Am einfachsten ist es, allen Code zwischen begin und end herauszulöschen und anschließend auf den Speichern-Button zu klicken. Delphi entfernt nun alle Methoden, die keinen Code enthalten.

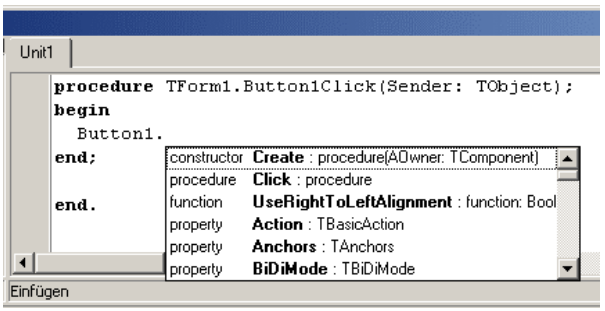
Wenn Sie ein ähnliches Beispiel Schritt für Schritt nachvollziehen wollen, verwenden Sie unser Tutorial "Mein erstes Delphi-Programm".

Editor

Mit dem Quelltexteditor von Delphi hatten wir bereits beim Erstellen von Ereignisbehandlungsmethoden zu tun.

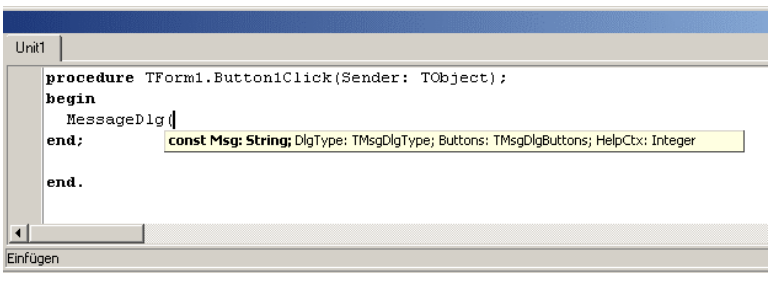
Bei dem Editor handelt es sich jedoch nicht um ein einfaches Texteingabefeld wie z. B. Notepad. Das Auffälligste ist, dass Delphi-Schlüsselwörter (wie begin und end) automatisch fett dargestellt werden, so dass der Überblick leichter fällt.

Code-Vervollständigung



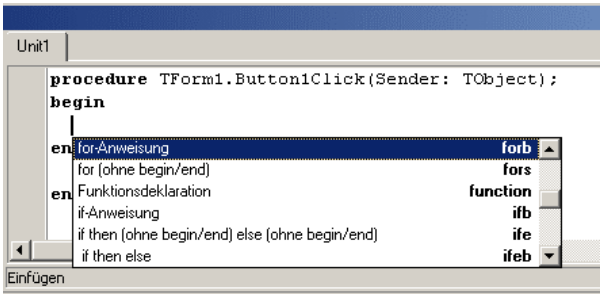
Es gibt weitere Elemente, die das Programmieren wesentlich erleichtern, z. B. die Code-Vervollständigung. Wird der Name eines Objekts eingegeben und anschließend der Punkt gedrückt, erscheint nach kurzer Zeit an dieser Stelle eine Listbox mit allen möglichen Eigenschaften oder Methoden, die nun mit Pfeiltasten und Return ausgewählt werden können.
Erscheint dieses Fenster nicht, erhält man es durch die Tastenkombination Strg+Leertaste.

Code-Parameter

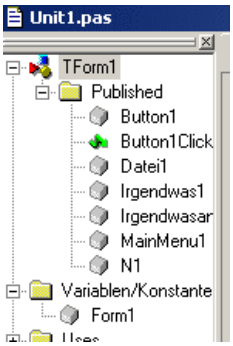


Auch eine große Hilfe ist die Anzeige der erwarteten Code-Parameter. So müssen Sie sich nicht die Reihenfolge und Typen von Argumenten merken, die ein Befehl erwartet. Jedes Mal, wenn nach einem Prozedur-/Funktionsaufruf eine öffnende Klammer eingegeben wird, erscheint der kleine Hinweis. Der aktuell einzugebende Parameter wird dabei fett hervorgehoben.
Erscheint dieser Hinweis nicht, drücken Sie die Tastenkombination Strg+Shift+Leertaste.

Code-Schablonen



Des Weiteren ist es auch möglich, häufig benötigte Code-Stücke als Schablone zu speichern. Durch Drücken der Tastenkombination Strg+J erscheint nebenstehendes Fenster, aus dem Sie einen Eintrag mit den Pfeiltasten und Return auswählen.
Über das Menü Tools/Editor-Optionen/Programmierhilfe können Sie weitere Code-Schablonen festlegen. Kleiner Trick dabei: Setzen Sie in dem Code-Stück an der Stelle, an der nach Einfügen der Schablone der Cursor stehen soll, ein Pipe-Zeichen ("|", AltGr+<).



Navigation im Code

Um sich in größeren Units schneller zu bewegen, gibt es mehrere Möglichkeiten: Zum einen ist die Navigation über den Code-Explorer möglich, der über das Menü Ansicht/Code-Explorer eingeblendet werden kann.
Außerdem ist es möglich, wenn man sich im Interface-Teil einer Unit mit dem Cursor in einer Methodendefinitionszeile befindet, durch die Tasten "Strg+Shift+Pfeil nach unten" direkt in den Implementations-Teil dieser Methode zu springen. Über "Strg+Shift+Pfeil nach oben" gelangt man wieder zurück.
Findet man im Code einen Befehl, dessen Implementation man sehen will, drückt man die Strg-Taste und fährt mit der Maus über dieses Wort. Daraufhin verwandelt es sich in einen Link, auf den man klicken kann. Schon befindet man sich an der gewünschten Stelle. Durch die Pfeile in der oberen rechten Ecke des Editors gelangt man wieder zum Ausgangsort zurück.

Klassenvervollständigung

Wurde im Interface-Teil einer Unit eine Klasse definiert oder einer bestehenden neue Methoden hinzugefügt, so genügt es, die Tastenkombination Strg+Shift+C zu drücken, um für alle noch nicht implementierten Methoden ein Quelltext-Grundgerüst (Kopf der Methode sowie begin und end) in den implementation-Abschnitt einzufügen.

Positionsmarken

Und schließlich ist es auch möglich, über die rechte Maustaste zehn verschiedene Positionsmarken im Code zu setzen ("Positionsmarken umschalten"), um bestimmte Stellen schnell wieder finden zu können ("Zu Positionsmarken gehen"). Zu beachten ist allerdings, dass diese nicht gespeichert werden und deshalb beim Neuladen eines Projekts nicht mehr verfügbar sind.

Suche

Wie in jeder ordentlichen Textverarbeitung ist es auch in Delphi über das Menü Suchen möglich, bestimmte Textstellen zu finden. Über "In Dateien suchen" lässt sich diese Suche auch über mehrere Units ausdehnen.

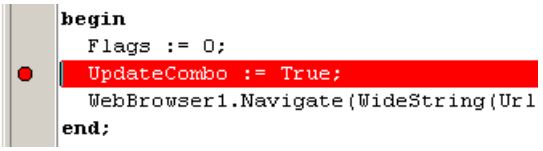
Debugger

Der Debugger ist eines der wichtigsten Hilfsmittel beim Programmieren. Denn wo Menschen am Werk sind, geschehen zwangsläufig Fehler. Je komplexer eine Anwendung ist, desto schieriger lassen sich solche Fehler finden.

Fehler, die auf einfachem Vertippen beruhen, bemängelt bereits der Compiler und bricht deshalb das Kompilieren ab. Fehler, die es bis ins laufende Programm schaffen, sind also umso schwer wiegender. Es handelt sich dabei entweder um Fehler, die das Laufzeitsystem zur Ausgabe einer Fehlermeldung bewegen (z. B. Variablenüberlauf, Typumwandlungsfehler), oder um etwas, das den Computer überhaupt nicht stört - Sie als Programmierer jedoch umso mehr, wenn das Programm nämlich nicht das Gewünschte tut. Das sind die logischen Fehler.

Bei der Suche nach den letzten beiden Fehlertypen hilft keine Syntaxprüfung, hier muss der Debugger ran. Und Delphi bietet einige Möglichkeiten, Fehlern auf die Spur zu kommen.

Haltepunkte



Das Erste, was zu tun ist, ist das Setzen von Haltepunkten. Diese wirken sich nur aus, wenn das Programm aus der Entwicklungsumgebung heraus gestartet wird (F9).
Gesetzt werden Haltepunkte durch Klick auf den grauen Randstreifen links neben dem Quellcode oder durch Drücken von F5.
Wird beim Programmablauf diese Stelle erreicht, bleibt das Programm stehen und wechselt in die IDE. Sie haben nun mehrere Möglichkeiten, wie Sie vorgehen können:

Variablenüberprüfung

Wenn Sie die Werte einzelner Variablen überprüfen wollen, genügt es, mit dem Mauszeiger darüber zu fahren. Es wird ein Hinweisfeld mit dem Wert eingeblendet. Hierbei ist wichtig zu beachten, dass dieser Wert nicht unbedingt den Wert darstellt, den die Variable an dieser Stelle hatte. Sondern es wird immer nur der augenblickliche Zustand wiedergegeben, also welcher Wert eine Variable zur aktuellen Ausführungsposition hat.

Sollen alle lokalen Variablen überwacht werden, können Sie über Ansicht/Debug-Fenster/Lokale Variablen ein Fenster anzeigen lassen, in dem die lokalen Variablen mit ihrem Wert aufgeführt sind. Dieses Fenster ist "andockbar", d.h. Sie können es an einem Rand des Editors "einklinken".

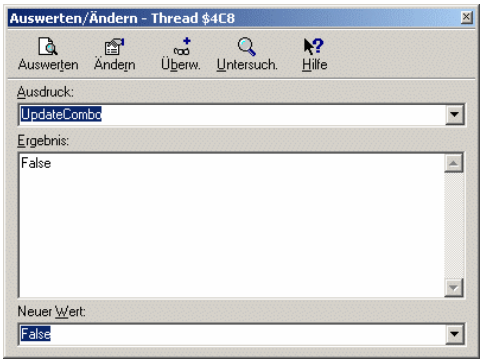
Wichtig ist auch noch das Fenster "Überwachte Ausdrücke", das ebenfalls über das Menü Ansicht/Debug-Fenster angezeigt werden kann. Hier können nun beliebige Variablen/Ausdrücke eingegeben werden (rechte Maustaste/Ausdruck hinzufügen), die überwacht werden sollen. Sie haben es also immer im Blick, wenn sich die Variablenwerte verändern.

Schrittweise Ausführung

Das Beobachten von Variablenwerten ist nur interessant, wenn sich die Ausführungsposition ändert. Momentan befinden wir uns ja noch an der Stelle im Code, wo Haltepunkt aktiviert wurde. Drücken Sie nun die Taste F7 (entspricht dem Menüpunkt Start/Einzelne Anweisung). Es wird nun genau die Anweisung ausgeführt, die durch den kleinen Pfeil und den farbigen Balken markiert war. Und sofort springt die Ausführungsposition zur nächsten Codezeile. Dies ist schon allein interessant, um zu sehen, ob ein bestimmter Programmteil (Bedingung, Schleife usw.) überhaupt jemals ausgeführt wird. Neben F7 gibt es nun noch F8 (Start/Gesamte Routine). Der Unterschied ist folgender: Trifft man mit F7 auf einen Methodenaufruf, dessen Implementation vorhanden ist, springt die Ausführungsposition dorthin. Es werden also alle auf dem Weg liegenden Methoden bis zur kleinsten Anweisung abgearbeitet. Mit F8 wird eine Methode dagegen nicht zwischendurch verlassen. Hier werden alle Anweisungen "am Stück" ausgeführt und nicht vom Weg abgeschweift. Beides kann in verschiedenen

Fällen nützlich sein.

Verändern von Variablenwerten



In Delphi ist es nicht möglich, im Debug-Modus die Ausführungsposition zu ändern, also z. B. eine Methode noch einmal auszuführen. Es ist jedoch möglich, Variablen andere Werte zuzuweisen als sie in der Programmsituation eigentlich hätten. Dies geht über den Dialog Start/Auswerten/Ändern. Unter "Ausdruck" muss der Name einer Variablen (oder eben ein Ausdruck) eingegeben werden, woraufhin unten der aktuelle Wert angezeigt wird. In der untersten Zeile kann ein neuer Wert eingegeben und über den Button "Ändern" zugewiesen werden.
So lässt sich prüfen, ob das Programm bessere Ergebnisse bringt, wenn eine Variable einen anderen Wert hat.

Aufrufstack

Will man sehen, welche Methode gerade welche Methode gerade welche Methode ... aufgerufen hat (wie diese Aufrufe also gerade geschachtelt sind), so erhält man diese Informationen über "Ansicht/Debug-Fenster/Aufruf-Stack".

Haltepunkt-Eigenschaften

Den Haltepunkt, den wir am Anfang verwendet haben, ist natürlich nur das simpelste Beispiel. Klickt man jedoch mit der rechten Maustaste auf den roten Punkt und wählt "Haltepunkt-Eigenschaften", lässt sich genau festlegen, wann ein Haltepunkt wirken soll. Beispielsweise lässt sich eine Bedingung eingeben (nur anhalten, wenn x>0) oder ein Durchlaufzähler einrichten (erst anhalten, wenn der Haltepunkt dreimal passiert wurde).

Man sieht also, dass Delphi sehr viele Möglichkeiten bietet, Fehlern auf die Spur zu kommen.
Und ist ein Programm einmal in einer so schlechten Lage, dass man es nicht fertig ausführen will, gibt es Strg+F2, das die Programmausführung unsanft beendet und (meistens) zu Delphi zurückkehrt. Unter zartbesaiteten Win9x-Betriebssystemen kann es unter Umständen hierbei (oder beim häufigen Anwenden dieser Funktion) zum Absturz kommen.

Komponenten und Packages

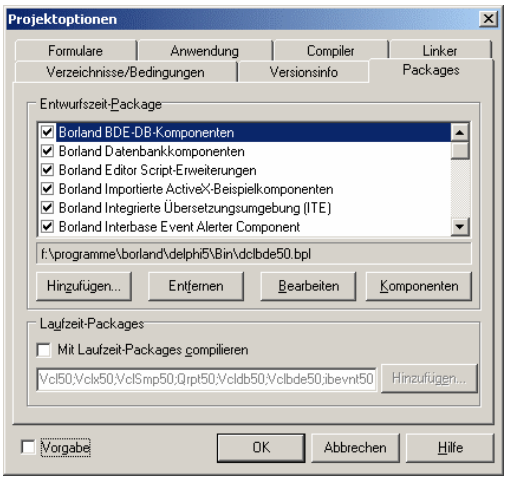
Mit einer Vielzahl von größtenteils kostenlosen Komponenten von den einschlägigen Komponenten-Sites lässt sich Delphi und somit die damit erstellten Programme um etliche Funktionen erweitern.

In der Regel handelt es sich bei diesen Komponenten aus dem Internet um Quellcodedateien, die in Delphi installiert werden müssen. Wie das funktioniert, wird im Folgenden beschrieben.
Liegt eine Komponente lediglich als kompilierte Datei vor (dcu), lässt sie sich - wenn überhaupt - nur unter der Delphi-Version installieren, für die sie kompiliert wurde. Sind die Quelldateien vorhanden, sind die Chancen einer erfolgreichen Installation also wesentlich größer.

Packages

Da man gleich zu Beginn einer Komponenteninstallation dem Begriff "Packages" begegnet, wollen wir zuerst klären, worum es sich dabei handelt.
Grundsätzlich ist ein Package eine Sammlung von einer oder meist mehreren Dateien. Die VCL-Komponenten, die Delphi beiliegen, sind bereits je nach Zusammengehörigkeit auf mehrere solcher Packages verteilt. Installiert der Anwender von Delphi zusätzliche Komponenten von Fremdherstellern, so kann er sie ebenfalls nach Belieben einem Package zuordnen oder ein neues erstellen. Wer Delphi von Anfang an kennt, weiß, dass es früher nur eine einzige Datei gab, in der alle installierten Komponenten versammelt waren. Kam es aufgrund einer fehlerhaften Komponente oder einer fehlgeschlagenen Komponenteninstallation zu Problemen, konnte die komplette Datei - und damit

auch Delphi - nicht mehr geladen werden. Borland hat dem Abhilfe geschaffen. Tritt ein Problem auf, wird nur das betroffene Package in Mitleidenschaft gezogen. Die übrigen können weiterverwendet werden.



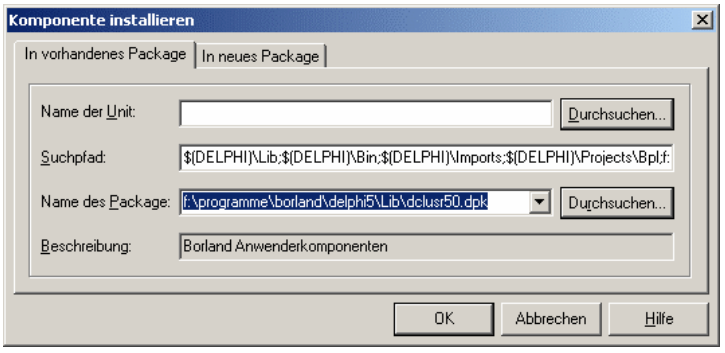
Bei Packages unterscheidet man zwischen Entwurfszeit- und Laufzeitpackages. Über das Menü Projekt/Optionen erhält man nebenstehenden Dialog. In der oberen Hälfte sieht man alle zurzeit installierten **Entwurfszeitpackages**. Wird ein Eintrag aus dieser Liste markiert und auf den Button "Komponenten" geklickt, erscheint ein Fenster, das anzeigt, welche einzelnen Komponenten dieses Package enthält. Entfern man einen Haken vor einem Packagenamen oder wählt "Entfernen", verschwinden die darin enthaltenen Komponenten aus der Komponentenpalette.
Die **Laufzeitpackages** sind eine Möglichkeit wie sie von Visual Basic her bekannt ist. In mehreren Programmen genutzte Code-Elemente (hier eben die Komponentenquellcodes) werden in eine oder mehrere externe Dateien (DLLs mit der Endung BPL) ausgelagert. Vorteil: Die eigentliche EXE ist wesentlich kleiner, mehrere (Delphi-)Programme können sich die Packages teilen. Nicht zu unterschätzender Nachteil bei der Weitergabe einer Anwendung mit Laufzeitpackages ist jedoch, dass diese sich von Delphi-Version zu Delphi-Version unterscheiden, was zu Konflikten beim Anwender führen kann.
Um Laufzeitpackages zu verwenden, genügt es, in den Projektoptionen den Haken zu setzen und alle in dem darunter liegenden Textfeld aufgeführten Dateien (Endung bpl) mit seiner Anwendung weiterzugeben. Die bpl-Dateien müssen im Windows-System-Verzeichnis untergebracht werden.

Folgende Datei-Endungen kommen durch Packages ins Spiel:

DPK	**Delphi Package** Hierbei handelt es sich um den Quellcode einer Komponentensammlung mit Verweisen auf die einzelnen Komponentenunits. Diese Dateien können mit Delphi geöffnet werden. Es erscheint der Package-Editor, der es ermöglicht, Komponenten aus Packages zu entfernen oder hinzuzufügen.
DCP	**Delphi Compiled Package** Wird ein Delphi-Package (dpk) kompiliert, erhält man eine dcp-Datei. Sie enthält einen Package-Header sowie die Verkettung aller Komponenten-DCUs. Der Name entspricht (bis auf die Endung) der dpk-Datei.
BPL	**Borland Package Library** Heißt in älteren Delphi-Versionen dpl (Delphi Package Library). Hierbei handelt es sich um eine Windows-DLL mit Delphi-spezifischen Eigenschaften. Diese Datei stellt ein Laufzeitpackage dar. Der Dateiname entspricht ebenfalls der dpk-Datei

Installation von Komponenten

Kommen wir nun zum eigentlichen Installieren von Komponenten.
Dazu wählen wir den Menüpunkt "Komponente/Komponente installieren". Daraufhin erscheint ein Fenster mit zwei Registerseiten. Wollen wir die



Komponente in ein bereits bestehendes Package einfügen bleiben wir auf der vorderen Seite, wählen im obersten Eingabefeld den Namen der Komponenten-Quelldatei sowie unten den Namen des Packages, dem die Komponente hinzugefügt werden soll.
Soll die Komponente in ein neues Package eingefügt werden, wechseln wir in die hintere Registerseite. Neben dem Namen der Komponenten-Quelldatei können wir hier nun einen Dateinamen für das neue Package sowie eine Beschreibung dafür eingeben.

Egal, welchen Weg wir gewählt haben, landen wir nun in dem bereits vorher erwähnten **Package-Editor**. Wollen wir gleich noch weitere Komponenten installieren, so können wir das hier über "Hinzufügen" erledigen.
Enthält das Package die gewünschten Komponenten, wird auf "Compilieren" geklickt, um aus DPK-Datei und den Komponenten eine DCP- und eine BPL-Datei zu erzeugen, welche sich anschließend im Unterverzeichnis Projects/BPL des Delphi-Verzeichnisses wiederfinden.
Durch Klick auf "Installieren" werden alle enthaltenen Komponenten in die Komponentenpalette aufgenommen. Ist dieser Vorgang erfolgreich, erscheint ein entsprechendes Meldungsfenster.

Da sich neue Komponenten häufig nicht dort in der Palette wiederfinden, wo sie evtl. sinnvoll sind, lässt sich über das Menü "Komponente/Palette konfigurieren", festlegen, aus welchen Registerseiten die Komponentenpalette besteht und welche Komponente sich auf welcher Seite befindet.

Das **Erstellen eigener Komponenten** soll nicht Thema dieses Grundlagen-Artikels sein. Lesen Sie dazu bitte das entsprechende Tutorial.