Whitepaper

An Introduction to Embarcadero® C++Builder® 2010

Volker Hillmann, adecc Systemhaus GmbH,

February 2009

Corporate Headquarters
100 California Street, 12th Floor
San Francisco, California 94111

EMEA Headquarters
York House
18 York Road
Maidenhead, Berkshire
SL6 1SF, United Kingdom

Asia-Pacific Headquarters
L7. 313 La Trobe Street
Melbourne VIC 3000
Australia

CONTENTS

EMBARCADERO C++BUILDER 2010

Embarcadero® C++Builder® 2010 is one of the leading, integrated RAD C/C++ development environments for generating native applications under Microsoft® Windows. This unique development environment combines the considerable flexibility of a RAD environment with the efficiency of the ISO-standardised programming languages C and C++. As a result, the leading programming language C/C++, which has already been designed to cover several paradigms, has been further extended. The current compiler already supports the large number of properties of the new ISO C++ standard C++0x which is being adopted in the coming months. Additionally, some of the Boost libraries will be supported directly, enabling C++Builder 2010 to connect more powerfully with the C/C++ community.

The application area of C++Builder 2010 ranges from fast prototyping to large-scale applications across all economic sectors, regardless of client or server programs. The supported target platforms range from Microsoft® Windows 2000 operating system through the current version of Microsoft® Windows 7 operating system. .

The integrated development environment (IDE) includes a powerful editor, which not only adapts to your individual habits as a developer, also supports syntax displays for the various file types, code templates, code completion, code folding, and automatic formatting. It is also equipped with a class browser, a visual designer for the user interface, an integrated help tool, and it is includes with an efficient debugger. The Project Manager, in which several applications and libraries can be jointly coordinated, includes wizards for creating special applications or objects as well as a history of the changes made to the source files.

The Designer provides the components of the supplied RAD framework VCL in the form of a Tool Palette so they can be easily inserted using a mouse. You can then edit the property values in a special area using the Object Inspector. The settings are saved parallel in a special file format instead of in the source text of the application.

C++Builder can be used to create various types of applications by selecting the application type from the New Items dialogue. Alongside the classic Microsoft Windows form application (GUI) you can also create text-orientated console applications or service applications. As the programming languages C and C++ play a key role in the creation of program libraries, both static and dynamic libraries can be generated using C++Builder.

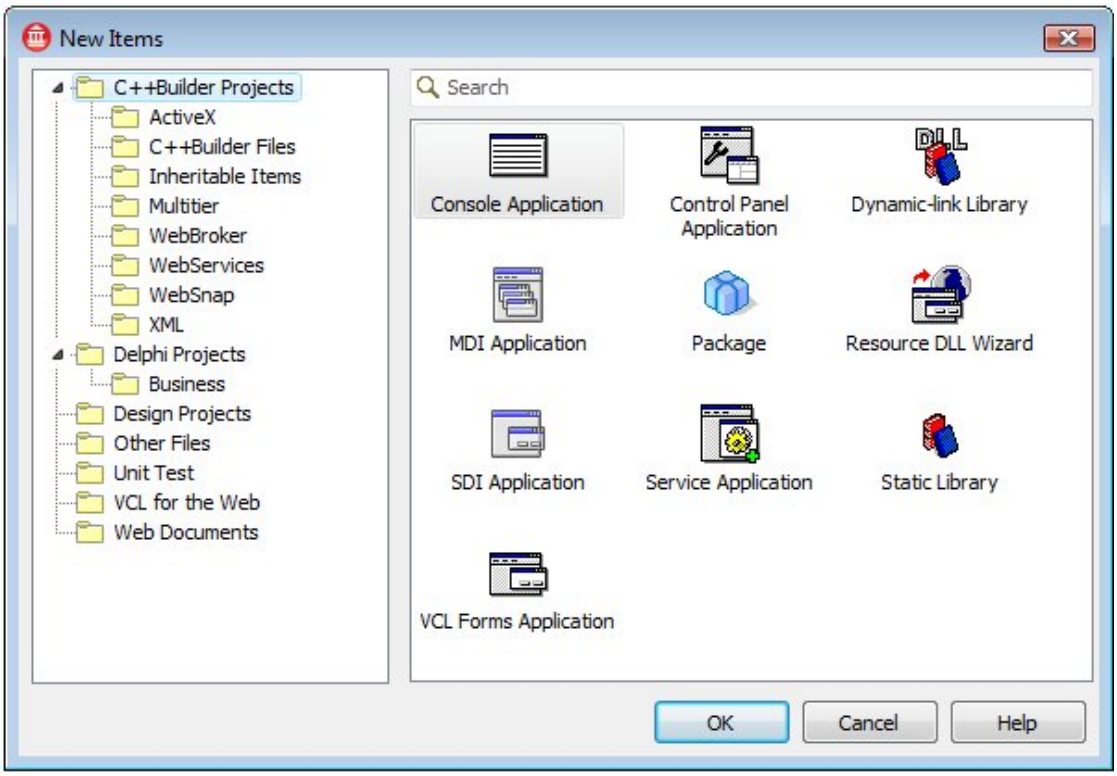

Figure 1: Selecting the target in the New Items dialogue

The New Items dialogue also includes the wizards which are used to generate special applications. With C++Builder you can also create applications for use with a web server, whether as a simple application or the implementation of a SOAP server. This also makes it possible to integrate in the infrastructure of large companies (SOA).

REASONS FOR C/C++ AS A LANGUAGE

C/C++ is a programming language, which, with the aid of a compiler, is translated into native machine code. This makes it possible to generate highly efficient code. While C remains rather restricted to system programming, C++ has been developed to become an universal language with a stronger, more static typification based on C, and directly supports multiple programming styles. Today, C++ combines the object-orientated with the procedural, the abstract and the generic programming. It is particularly the generic programming that enables a high degree of flexibility. In doing so, the programmer has the choice, and styles can be combined at will. As a conscious effort has been made to reject platform-specific properties, C++ is not just as fast as C, it is just as easy to port.

By contrast with many other programming languages, C++ is not owned by one company. The language was developed right from the start in collaboration with several companies, and has been standardized by ISO since 1998. Many companies and universities, including Adobe, Apple, Microsoft, IBM, Embarcadero, HP and Google, were involved in its further development.

The C++ concept enables both machine-orientated but also highly abstract programming. In the second case, the benefits lie in a considerable expressiveness and flexibility. A criticism often heard is the lack of free storage management. C++ does actually possess an adaptable free storage management in which you can seamlessly integrate an automatic garbage collector. This means that C++ is also implemented widely in the industrial sector and is very suitable for large-scale projects. The compatibility with C, previous lack of which has often been criticised, ensures a very rapid distribution.

C/C++ is widely used as a programming language in the UNIX field, and is therefore also available for Linux. With the GNU C/C++ compiler, many platforms can enjoy a very efficient implementation for the programming language in the open source area, therefore securing investments in C++ programs independently of commercial companies.

THE ISO STANDARD

The programming language has been standardised under the designation "ISO/IEC JTC1 SC22 WG21". One of the great advantages of C++ is the open standard and the vast number of people, companies and universities involved in its further development. This enables a continuous and practical advancement of the language.

The first standard was agreed in 1998. After a small step in 2003, the new standard, C++0x, will be concluded in the coming months.

The members of the Committee originate from the areas of scientific research and industry. Herb Sutter, one of the software architects from Microsoft and known from the "Guru of the Week" series, recently became the chairman of the Committee. This is a very clear indication – Microsoft marketing backs C#. The most important products are mainly written in C and C++. If you trace the names of the Standards Committee members, you will discover, for instance, companies like HP, Apple, Google, IBM, Embarcadero, Adobe, Intel, Oracle (Sun), Red Hat, SGI, AT&T, … . Some of the greatest universities in the USA, like the Texas A&M University, the Indiana University and the Washington University, are also involved in its further development. Added to this are major library and tool manufacturers, e.g. boost, Rogue Wave and Dinkumware.

THE DEVELOPMENT ENVIRONMENT

The individual parts of the program are linked to each other within the development environment. When you launch C++Builder, it appears in the default layout. By contrast with the older versions 5 and 6, the individual parts are combined (docked) in one main window.

The following figure shows the default layout of the development environment.

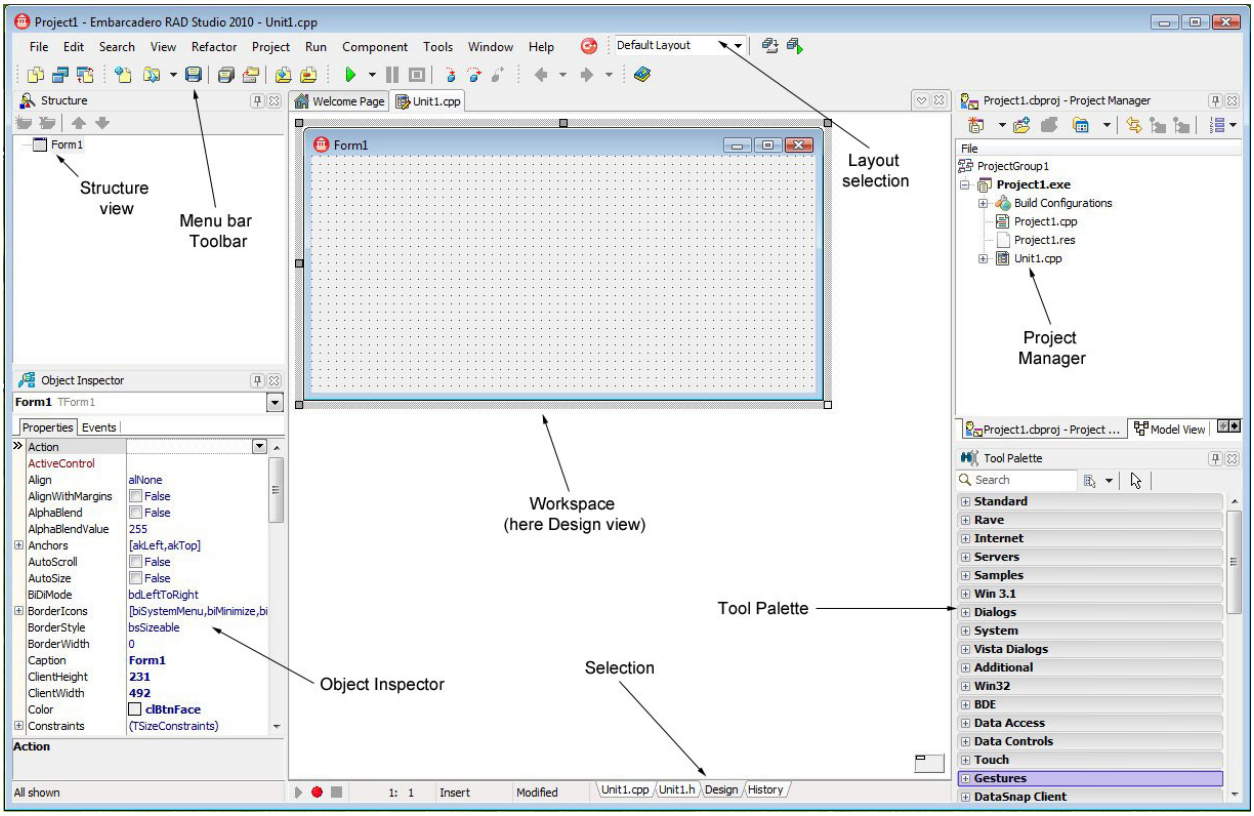

Figure 2: Development environment in the default layout

The main menu and toolbar are located in the upper area of the application, the workspace in the central area, and additional windows to the left and right. The workspace features an embedded Form Designer in which forms can be edited visually. The Structure view, the Object Inspector and the Tool Palette allow you to work on the form with VCL components, and these are arranged around the workspace.

The lower area of the workspace includes a selection area. Here you toggle between the form view, the declaration and the implementation. You can edit the source text in the editor by using the syntax support, code completion and integrated tooltips.

As the majority of the applications are made up of more than one source text file and libraries, a further part of the environment, the Project Manager, is used to coordinate the associated source files and libraries.

With C++Builder 2010 you can return to the classic view as is customary in older versions (Builder 5 + 6). Many developers have had problems with the changeover to the new default layout which has been used as of C++Builder 2006. This has a selection field alongside the menu bar. In the classic layout, the windows are once again loose (undocked) on the screen, and the Tool Palette is again located at the top right. This makes it easier for developers to switch from an older version to C++Builder 2010.

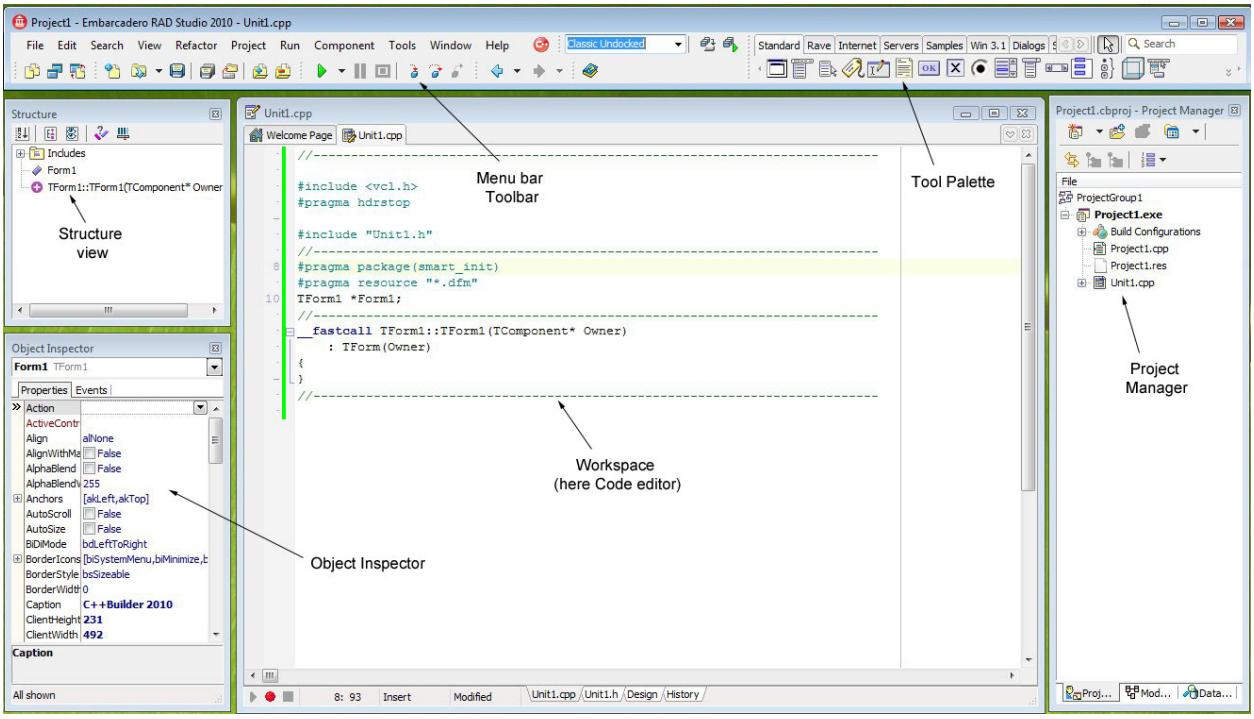

Figure 3: Development environment in the classic mode

As standard, the design interface is still fixed in the workspace - even in the classic layout. But this can be adapted. To do this, you select the area "Environment options" under "Tools > Options" and then go to the "VCL designer" page. Here you disable the option "Embedded designer" and then restart the development environment. This then undocks the design interface, as in the previous versions.

C++Builder 2010 helps you to search for new properties. Pressing the function key "F6" launches the IDE Insight wizard which then supports you in the search for options and properties.

IDE INSIGHT WIZARD

The "IDE Insight wizard" is a new feature in C++Builder 2010 development environment. This little tool can help you search for information in various categories incredibly quickly. The

searched text is entered in the input field and the search takes place incrementally. When the required property appears in the display range of the dialogue, you simply double click on this and the system jumps to the dialogue you are searching for or the respective command is executed.

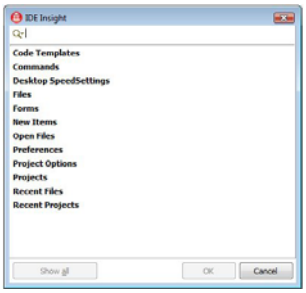

Figure 4: IDE insight

If, for example, you enter "console", and the entry "Console Application" appears, double clicking on this with the left mouse button launches the wizard for creating a console application.

You can find similar quick searches in other systems. The operating systems Microsoft® Windows Vista and Microsoft® Windows 7 have a similar help function under the Start button.

PROJECT MANAGER

This is the central point in which you work with projects and project groups. As standard, the Project Manager is located in the right section of the development environment. You can open only one project group at a time in C++Builder 2010, but several projects can be located within this group. A project group has the extension "*.groupproj".

A project contains all the information that is required to generate an application or a library. In addition, C++Builder 2010 saves all the source files, resources, libraries and settings for the compiler and the linker in the project file. This is written in an XML format and, in the current versions, has the extension "*.cbproj".

The following figure shows the Project Manager with two projects. The information is shown hierarchically.

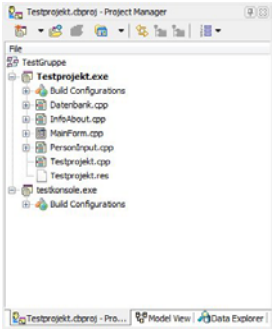

Figure 5: Project Manager

In this example, the project group "TestGruppe" has been opened. This contains the two projects "Testprojekt.exe" and "testkonsole.exe". The project "Testprojekt.exe" is enabled (shown in bold) and the nodes expanded.

In the Project Manager, you can generate new projects or add existing ones.

VIRTUAL FOLDERS

To achieve a better overview, you can set up virtual folders and simply drag & drop the files into these. There then follows a grouping according to contents, but the files are only displayed under this folder. A further advantage of the virtual folder is that you can display the files in this without the sometimes annoying physical path.

You can disable the view of the virtual folder without actually deleting it. If a virtual folder is deleted, the files still remain in the project.

SORTING THE DISPLAY

In the previous versions it was possible to change the order only by drag & drop. In doing so, there was another trick - the files that were being dragged with the mouse also had to be dragged to the left on the structure lines before they were saved. While some developers prefer an alphabetic sorting, others use an arrangement according to content.

This is why C++Builder 2010 now has the possibility of sorting the files in the view. You can now arrange the files according to the names, the date of update, the path, the type or the order of build. You can find the selection in the Project Manager toolbar or in the respective context menu.

The order of build plays a key role and can always be changed by drag & drop or the respective entries in the context menu of the Project Manager. C++Builder 2010 has a special view in which you can manage the order of build.

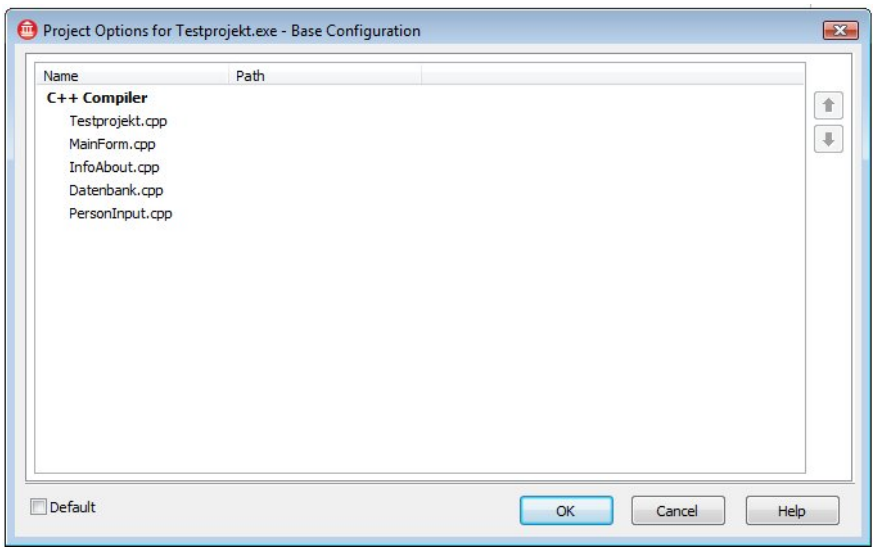

Figure 6: Managing the order of build in the Project Manager

SETTINGS

The settings play a very important role in creating the projects. Both the compiler and the linker are called as external programs. Even though the key principle of C++ is for developers to be able to work without a respective development environment, and programs and libraries are very often also generated in batch, it is difficult to keep specifying all these parameters. This is why C++Builder makes available a comfortable management of the options and saves this in the project files.

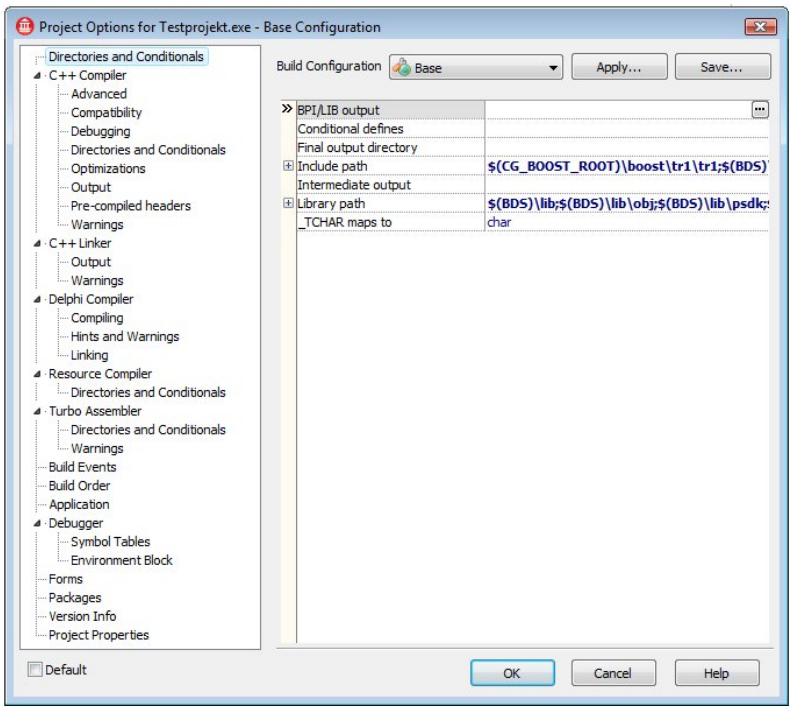

Figure 7: Project settings

If applications and libraries are created from the Project Manager, the development environment will then use these settings accordingly.

Careful: The selection box "Default" is located in the lower area of the dialogue. If this selection is enabled, all the settings are adopted in a template which is then used each time a new project is generated.

BUILD CONFIGURATIONS

In doing so, the settings are also hierarchically summarised in groups, the "build configurations". As standard, C++Builder 2010 recognises three groups, the key settings are saved in the base configuration, but deviating ones saved in the debug or release configurations. You can, however, set up other configuration sets. You can save these settings in project-independent files (option groups *.optset) and then use them again in other projects.

CREATING THE APPLICATIONS AND LIBRARIES, CLEANING

There are two variants to consider when creating applications and libraries. When generating a project, all the files are recompiled and then linked together at a later point. When updating, only the files that have been changed since the last generation / update are recompiled. And if only one file has been recompiled, the linker is then launched.

In the case of a cleaning, all the interim files are deleted, so they will have to be recompiled afterwards in any case.

THE "FROM HERE" MENU ITEM

The menu item "From here" is a new feature in the Project Manager. With this function you can regenerate, update or adjust all the projects following the one you have selected.

PREPROCESSOR, ASSEMBLER, MEMORY DUMP

Further areas which have been enhanced in the current version of C++Builder are the preprocessing (C preprocessor), assembling and the display of the memory dump of a module or an application. The first two areas already existed in the previous versions and are considered standard in a C/C++ compiler. They give the developer the option to control macros and further optimize the source code.

The memory dump doesn´t just show the assignment of the main memory, but also the methods and libraries used.

SOURCE TEXT EDITOR

The source text editor enables you to work directly with the program source text. While some parts of the source text are inserted and edited by C++Builder itself, others are edited manually by the developer. As you would expect from an efficient editor, there are various modes. Text can either be added or overwritten. Coloured markings on the border indicate whether this is changed source text or whether this has already been saved.

TEXT SEARCH

Searching in the source text window

As it is very important for a developer to find information quickly within the source text, you can search for text in the source text window. With C++Builder 2010 this option has been improved to include a search input area in the lower area of the source text window. Go to the "Find" entry under the "Search" menu to enable this area.

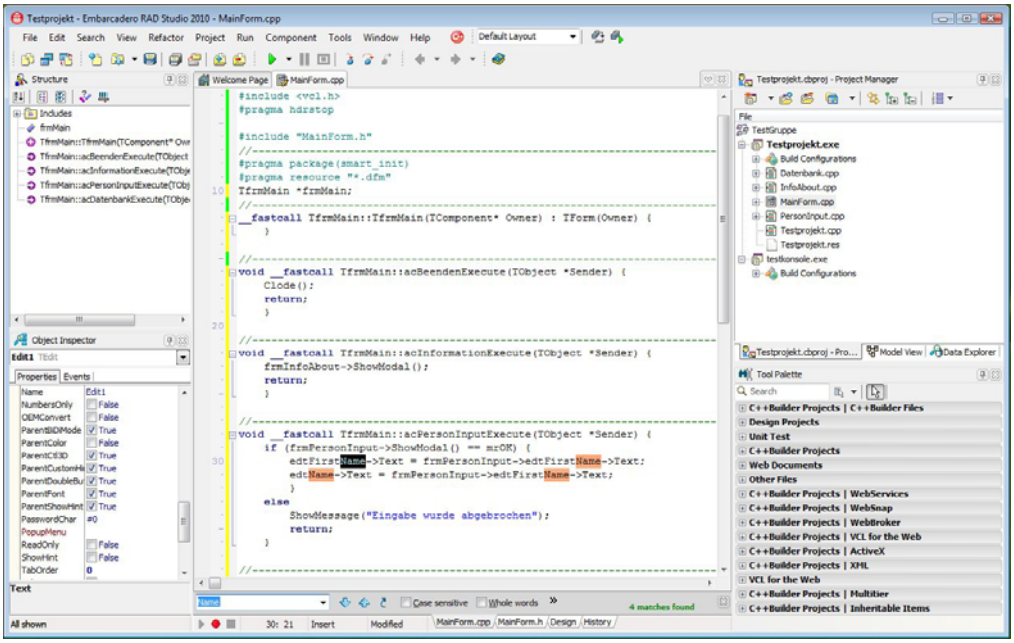

Figure 8: Searching in the source text window

The input field is a selection area in which you can display the last search query. As you can see in the figure, there are several search options. You can differentiate between uppercase and lowercase, search for whole words, search in the selected area and use regular expressions. The text references found are shown in colour and the number of matches is displayed in the lower area. Use the arrow keys to switch between the found matches.

As an alternative, you can also search incrementally, but this does not support the additional options.

Text search in projects and files

There are more options available in the "Find in Files..." entry in the "Search" menu. A modal dialogue opens.

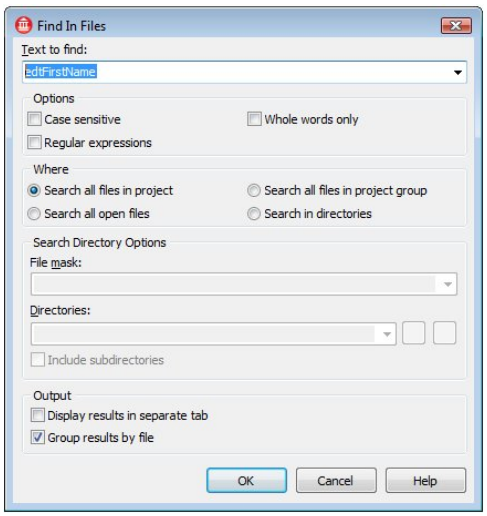

Figure 9: Find in Files

This selection enables you to search in all the files of the current project or project group but also only in the files which are currently open or in the specific directories. You can also use the options described above "Case sensitive", "Whole words only" and "Regular expressions" here. The search results are displayed in the message window and can be grouped according to files.

Searching and replacing text

It goes without saying that C++Builder gives you the option to search for text and replace this with other text. Go to the "Replace…" entry in the "Search" menu for this. A modal dialogue opens in which you can enter the text phrases and options.

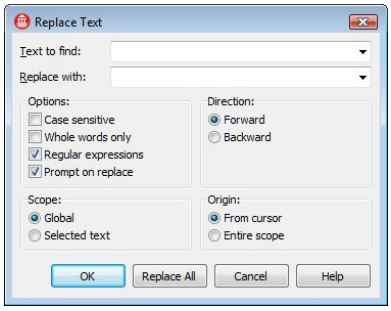

Figure 10: Replace Text

REFACTORING

Refactoring is the restructuring of the source text (structure improvement) whilst maintaining the behaviour. This should improve the maintainability and expandability of the programs. C++Builder 2010 supports this via the "Refactoring" area; you can search for references of variables and carry out automatic renaming. You can display all the respective positions prior to renaming.

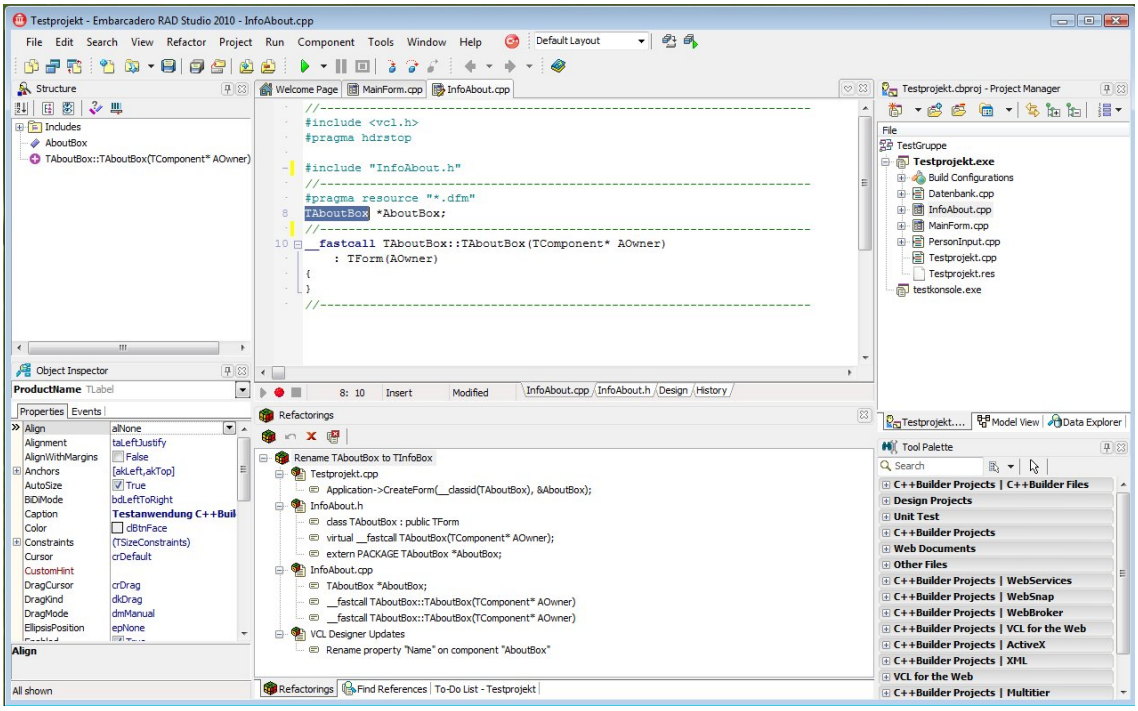

Figure 11: Refactoring

As many developers initially start with the automatic allocation of names for components, this option is very important, ensuring you can obtain a project in a legible form at a later date.

CODE COMPLETION AND PARAMETER SUPPORT

The source text window features some additional aids which aim to support the developer in writing source text. The most important are the code completion and the parameter support. These two services belong to the "Code Insight" area and can be switched on or off in the options accordingly. Moreover, you can regulate the delay before the support is enabled.

The "Code Insight" area includes other properties. For example, while debugging an application you can display variable content with a tooltip or automatically close brackets. In addition to this, you can control the behaviour of the code templates here.

Important: In larger projects it makes sense to disable some of the options in this area, as these could cause considerable delays when processing the source text.

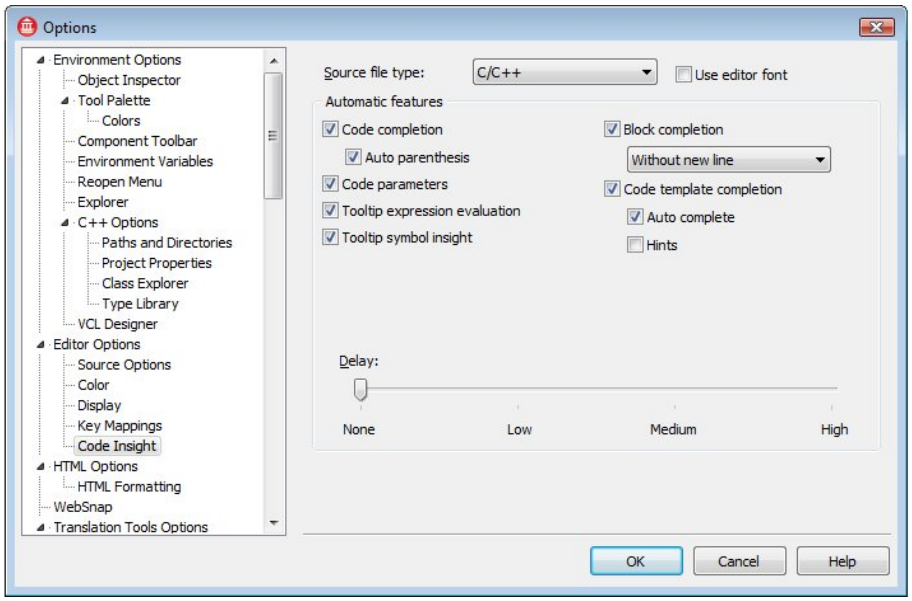

Figure 12: Options for "Code Insight"

With code completion, the development environment searches for known objects and properties and makes these available in a selection dialogue. These can be adopted immediately without having to enter the full identifier.

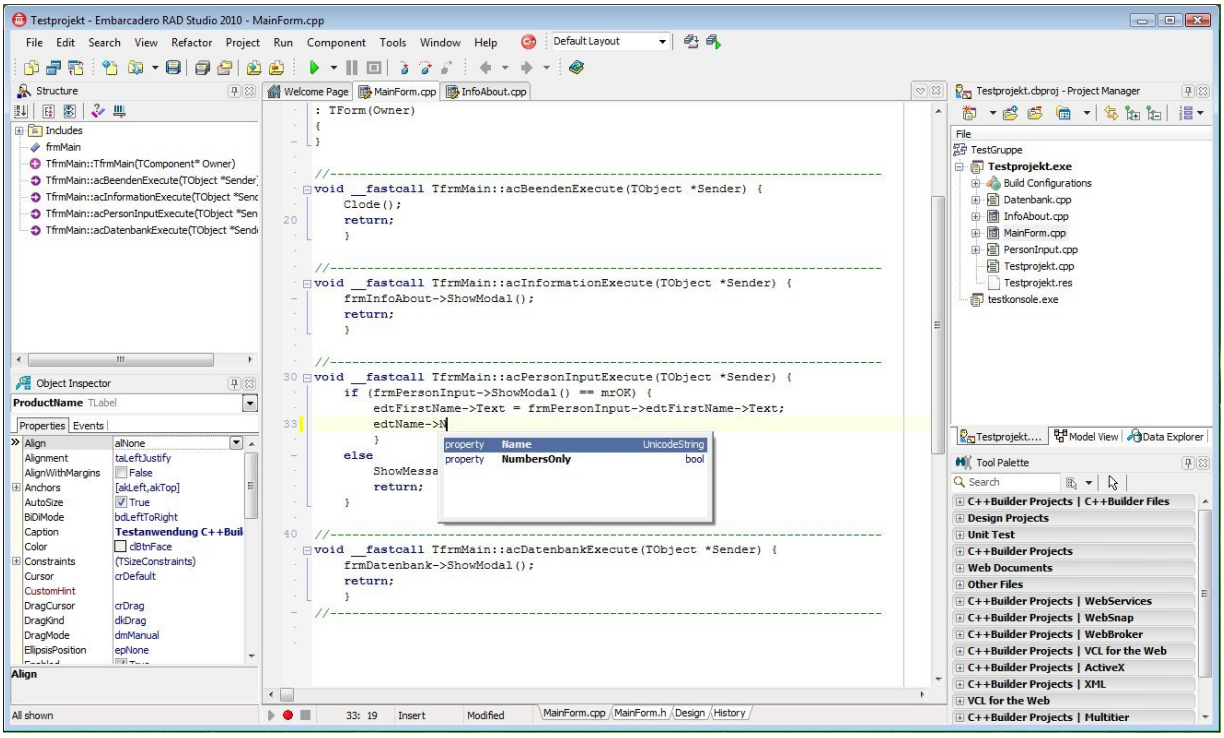

Figure 13: Code completion

With parameter help, the development environment displays the names and types of call parameters for the methods entered.

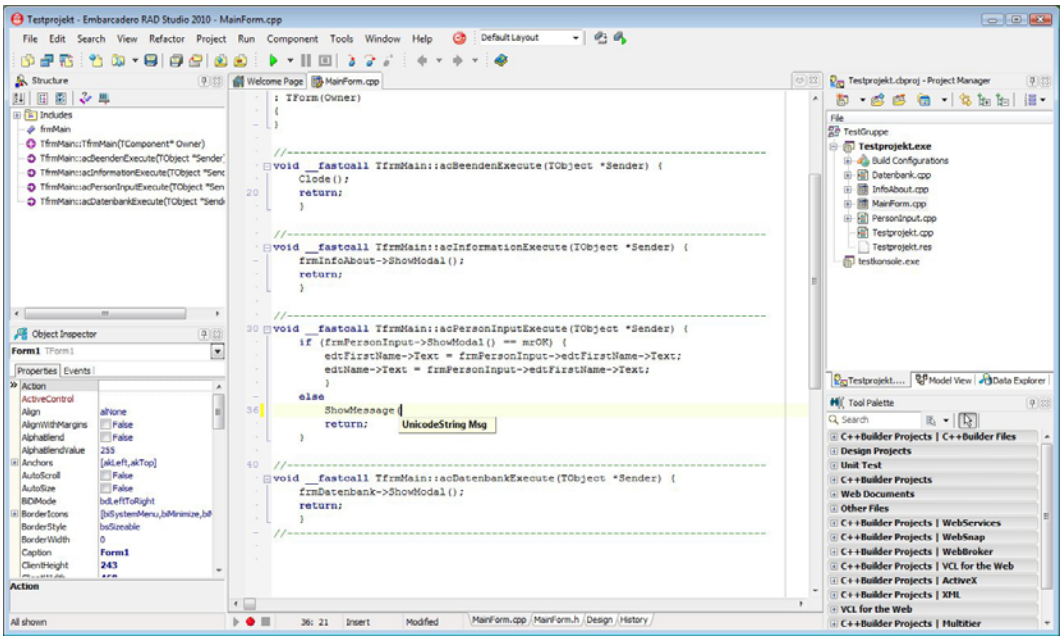

Figure 14: Parameter help

CODE FOLDING

As standard, the source text window offers the option of "folding together" the functions and classes so as to offer a better overview. However, C++Builder 2010 also gives you the option to add other regions and to provide these with a description. For example, you can bracket algorithms and then unfold or fold them. If a region is folded, the source text window shows only the framed description.

```
//===========================================================================
//  Example program C++Builder 2010
//  adecc Systemhaus GmbH
//  Copyright (c) 2008-2009
//===========================================================================
//  random and bind with regions
//===========================================================================

#pragma hdrstop
#include <tchar.h>

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <vector>
#include <random>
#include <functional>
#include <algorithm>
```

```
// Help functions for sorting
template <typename T>
bool is_less_than(T a, T b) {     return a < b; }


// Help class for output of figures in groups
class TOutputHlp {
   private:
      int iCount;
      int iMax;
   public:
      TOutputHlp(int para) : iCount(0), iMax(para) { };

      TOutputHlp(TOutputHlp const& ref) {
         iCount = ref.iCount;
         iMax   = ref.iMax;
         }

      virtual ~TOutputHlp(void) { };

      void Reset(void) {
         std::cout << std::endl;
         iCount = 0;
         return;
         }

      template <typename T> void operator () (T const& val) {
         std::cout << std::setw(3) << val;
         if(++iCount >= iMax) {
            iCount = 0;
            std::cout << std::endl;
            }
         return;
         }
   };

//---------------------------------------------------------------------------

#pragma argsused
int _tmain(int argc, _TCHAR* argv[]) {
   TOutputHlp output(10);
   std::vector<double> values;
   #pragma region dice algorithm
   std::tr1::minstd_rand      rand_gen(std::time(static_cast<std::time_t *>(0)));
   std::tr1::uniform_int<int> uniform distribution(0, 60);
   std::tr1::variate_generator<std::tr1::minstd_rand,std::tr1::uniform_int<int> >
                                       generateValue(rand_gen, uniform distribution);
   for(int i = 0; i < 40; i++) values.push_back(generateValue());
   #pragma end_region

   #pragma region output
   std::cout << "Output of generated random numbers" << std::endl;
   for_each(value.begin(), value.end(), output);
   output.Reset();
   #pragma end_region
```

```
#pragma region sort ascending and output
std::sort( value.begin(), value.end(), is_less_than<double>);
std::cout << "Sort ascending values" << std::endl;
for_each(value.begin(), value.end(), output);
output.Reset();
#pragma end_region

#pragma region sort descending and output
std::sort(value.begin(),value.end(),std::tr1::bind(is_less_than<double>,_2,_1);
std::cout << "Sort descending values" << std::endl;
for_each(value.begin(), value.end(), output);
output.Reset();
#pragma end_region

return 0;
}
```

Listing 1: Random numbers with code folding

If the region is visible, it is recognised by the character [-] in the border. The area is marked by a line. Otherwise, the [+] symbol precedes it.

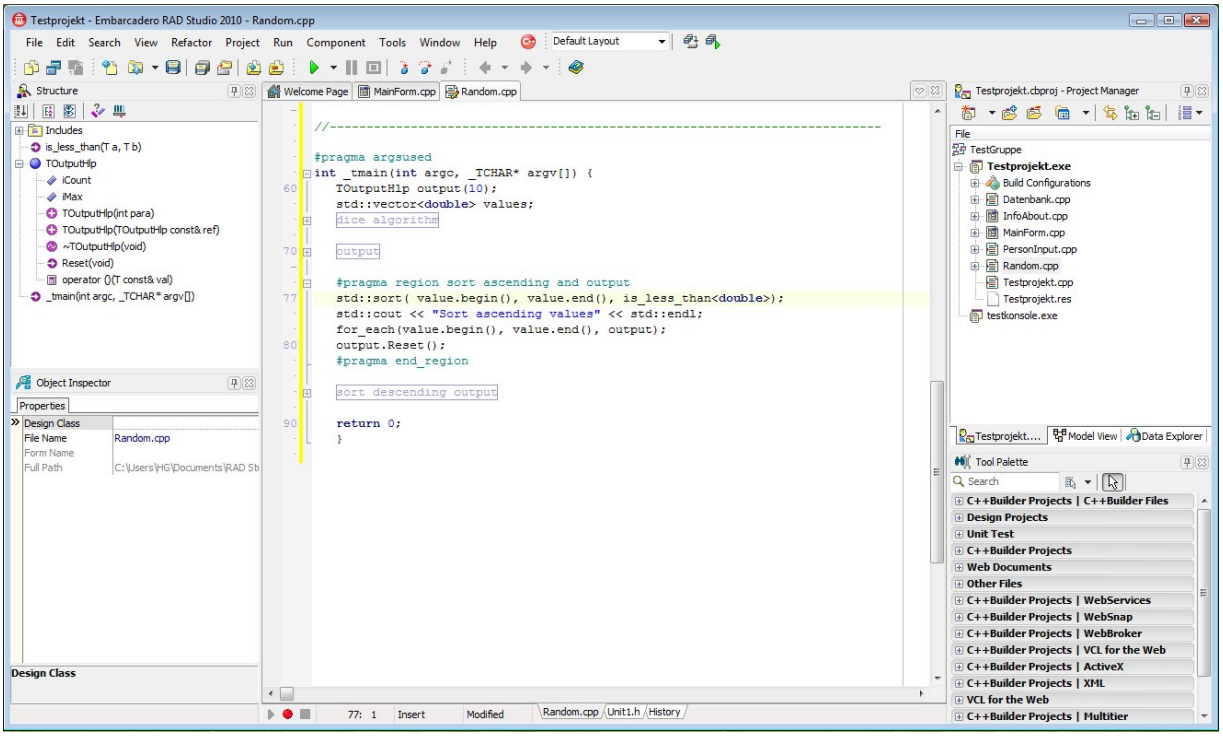

Figure 15: Regions in the source text window

As with the majority of the editor options, you can also switch off folding. Go to the "Code folding" checkbox under the "Editor Options" tab (or use the shortcut "Ctrl+Shift- K+O").

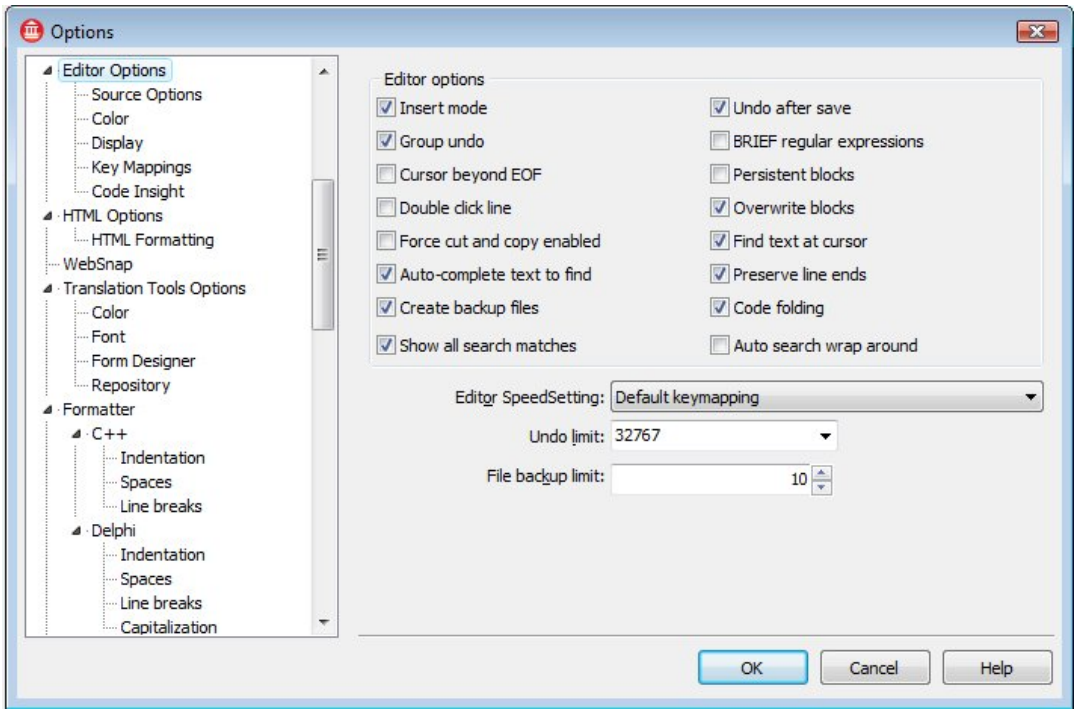

Figure 16: Editor Options

To hide a region, use the mouse to click on the [-] symbol or use the shortcut "Ctrl+Shift - K+E". To show, click on the [+] symbol or use the shortcut "Ctrl+Shift - K+U". You can also make all the hidden areas visible again by using the shortcut "Ctrl+Shift - K+A".

TEMPLATES

With code templates you can manage frequently recurring source text and accelerate the input. The possibilities have been further enhanced compared to the previous version. Today the templates are saved in an XML format and are available for the different source formats. You can add templates automatically ("Code Insight") or manually (shortcut Ctrl+J).

You work with templates via the" View" menu. Refer to the entry "Templates" in the "View" menu. The window used to edit the templates then appears at the Tool Palette position. Here you can see any already predefined templates.

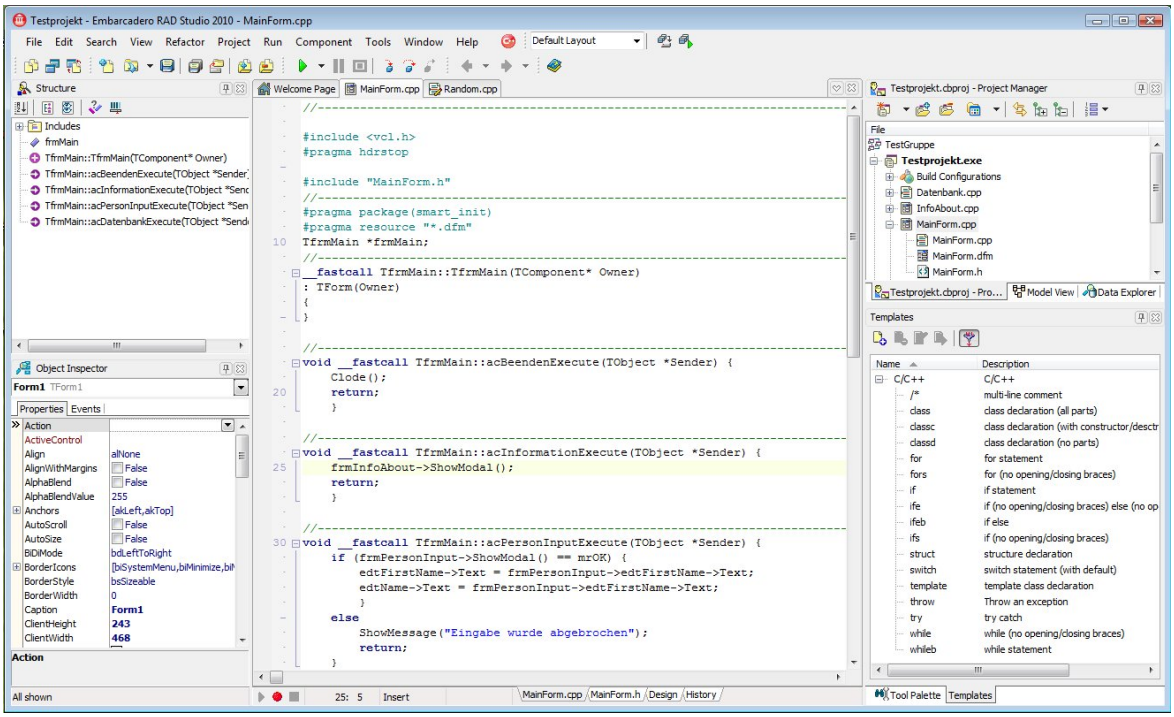

Figure 17: Development environment with code templates

To manually trigger an existing template, enter the name in the source text and then press the shortcut Ctrl+J. The source text is then added accordingly. If the name is incomplete or missing, a selection window appears showing the queried templates. As an alternative, you can double click with the left mouse button on the desired template in the template window; this is then adopted at the cursor position.

You can work with the templates via the toolbar or the context menu in the template window. Here you can create new templates, delete those which are no longer required or simply edit an existing one.

Creating new templates

To create a new template, select the respective icon in the toolbar of the template window or click on the "New" entry in the context menu. An XML file showing the basic structure for the new template appears in the code window.

By default, new templates are saved in the user's documents folder. Only here is it assured that the necessary authorizations for generating new files are available. To publish this template in the development environment, you must first copy the xml file to the subdirectory "ObjRepos\en\Code_Templates\c" of the respective installation directory of C++Builder 2010. The subdirectory "de" of the respective localisation (German here) and "c" corresponds to the language for which the template is being used.

In our example, we want to create a new template for the documentation of files in the format of "Doxygen". Doxygen is a project with which programs can be documented in C/C++. This is

distributed under the terms of the GNU- GPL and can be downloaded from the Internet. I have deliberately chosen this example, as we are always hearing about how complex the documentation is, which is why many developers will not use it.

The necessary information for this is located within the source text, so it is no longer necessary to synchronise between source code and documentation. To describe the information for a file, in a source file for Doxygen there is the following format.

```
/** @file
 * @brief file with definition of the class TSysTime for querying the system time
 * @author Volker Hillmann
 * @author adecc Systemhaus GmbH
 * @date 15.05.2006
 * @version Version 1.3
 * @since Version 1.3
 */
```

Listing 2: Doxygen comment for a file

Of course, there are further commands, e.g. to document errors and warnings or to manage a ToDo list. Refer to the Doxygen homepage for more information.

We often hear the criticism that the source text is often easy to overlook on account of the extensive documentation. Here, C++Builder offers the option of individual code folding, so that the areas are written in the regions and can be folded accordingly in order to edit the source text.

In doing so, you can save descriptions and notes and also define points which are then used in the code snippet of the template. The following figure shows the new template “DoxyFile”.

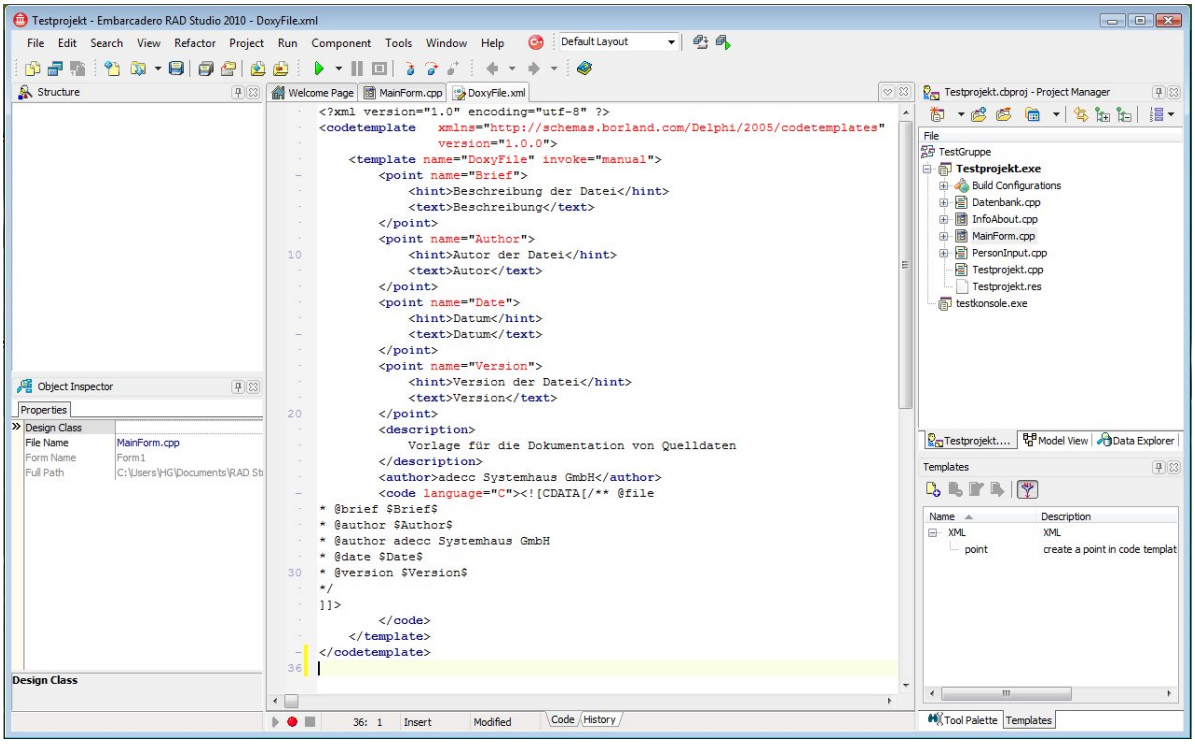

Figure 18: Editing code templates in the code window

If you now create a new unit and then enter the name “DoxyFile” followed by the shortcut “Ctrl+J”, “DoxyFile” is overwritten by the new template. The following figure illustrates this. The points entered in the template are framed and will be overwritten when entered. Provided you are within the inserted template, you can then use the tab key to toggle between points.

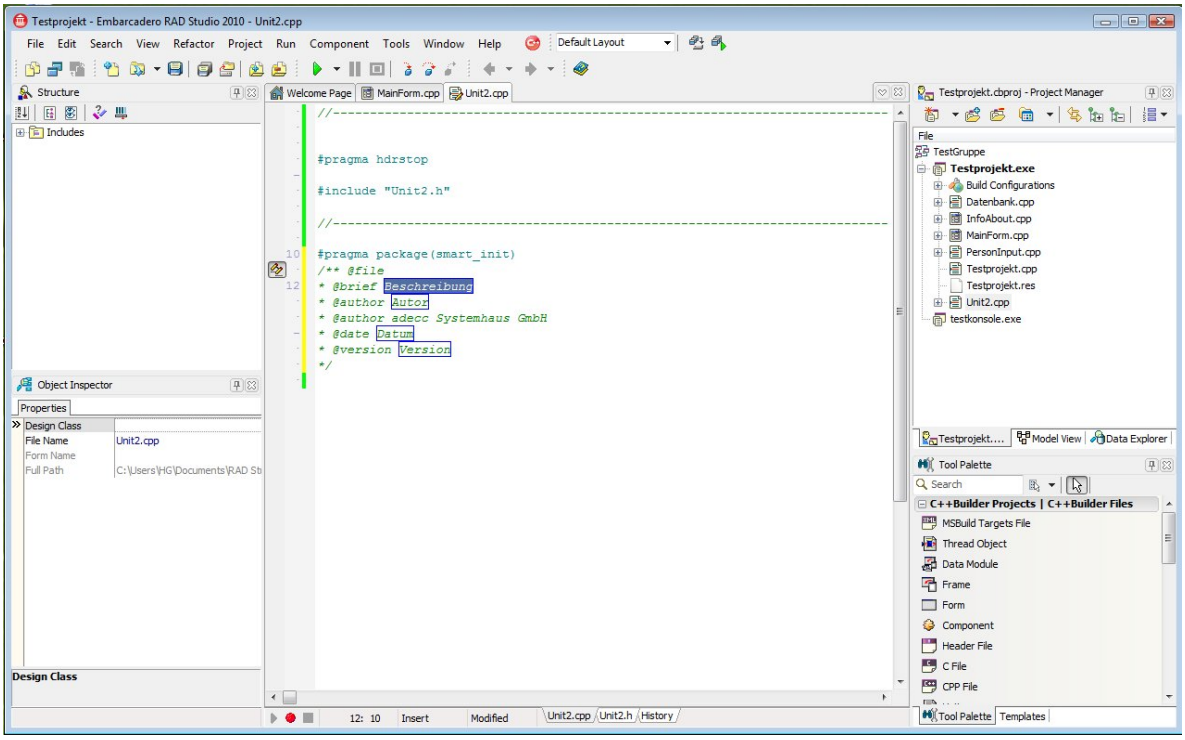

Figure 19: Inserting a code template

As the example shows, by using the code templates you can build a skeleton in order to save a lot of time when working with the source text and to introduce respective standards. As the templates are saved in xml files, these can be changed easily.

SOURCE CODE FORMATTING

Developer teams are divided in terms of the nature in which the source code can be best formatted. Some will write the opening brackets at the top, others always in the next line. Some developers leave a space between the identifiers and the operators, others, however, don´t.

C++Builder 2010 finally puts an end to this discussion. In the options you can establish how the source code should be formatted. For this purpose, the appropriate options are set in the areas “Indentation”, “Spaces” and “Line breaks”.

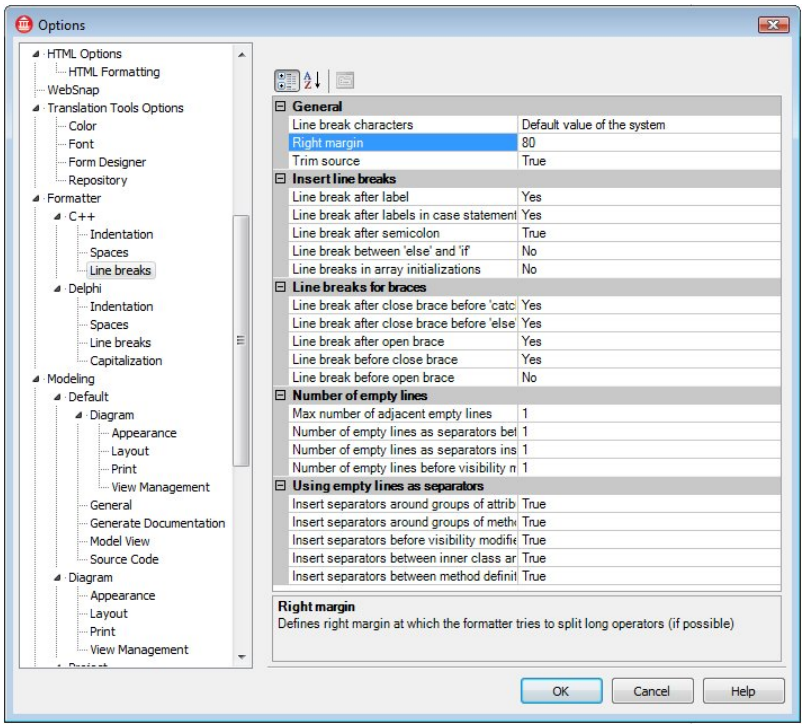

Figure 20: Options for code formatting

Now you can automatically format the source text via the context menu in the code window or with the shortcut "Ctrl+D". If only a specific area of the source text is selected, then only this selection will be edited by the automatic formatting.

C++ Class Explorer

The older versions of C++Builder 5 and 6 were provided with a Class Explorer. However, as this is an offshoot of the Delphi Class Explorer, and Delphi does not cover the possibilities of C++ class hierarchies, there were always problems when it came to complex applications. It was particularly difficult to solve the multiple inheritance. This is why the last versions did not include the Class Explorer, but this was requested by a lot of developers.

However, we see its return as a special C++ Class Explorer in C++Builder 2010.

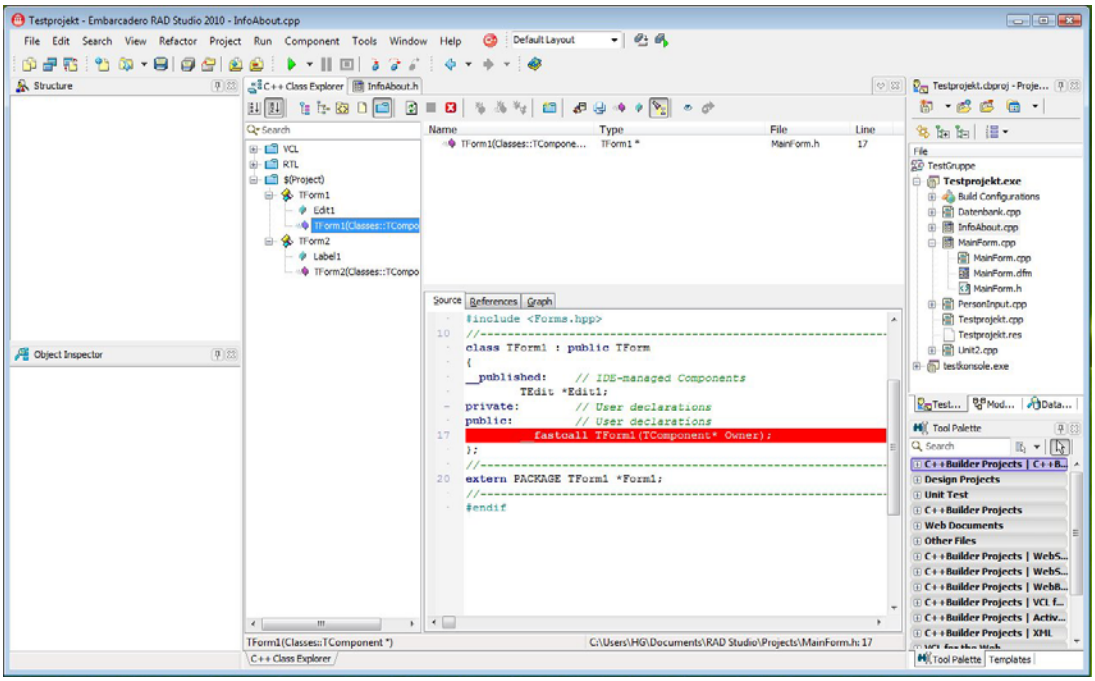

Figure 21: C++ Class Explorer – normal view

When the Class Explorer starts up, the project is scanned and the necessary information searched for. To enable a better overview, the classes are immediately subdivided into categories.

Category	Description
RTL	Classes, variables and functions of the runtime environment
VCL	Classes, variables and functions of the VCL (if available in the project)
STL	Classes, variables and functions of the STL (if available in the project)
Project	User-defined classes and variables in the project

Table 1: Categories in the C++ Class Explorer

You can also display enumerations (enum) and self-defined types (typedef) in the Class Explorer.

NAVIGATING IN THE SOURCE TEXT

If you click in the left area of the Class Explorer a class or variable, a view which displays the respective source text opens in the right, lower area of the Class Explorer. A list of the variables, properties and methods is shown in the right, upper area. In this list you can also see in which file and at which position the respective element can be found. The inherited properties are also shown here.

If you now click on the entries at the top, the view is synchronized in the right, lower area.

By using commands in the toolbar or the context menu you can now switch to the declaration or definition in the code window. It is then possible to edit as usual, so that with the aid of the Class Explorer you can very quickly navigate in the source text of the project.

ADDING NEW ELEMENTS TO A CLASS

You can also add new properties within the Class Explorer. The appropriate commands are available in the toolbar of the Class Explorer for this purpose. The following figure shows the wizard for adding a new method.

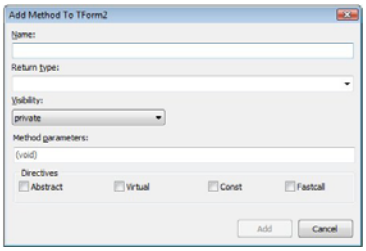

Figure 22: Adding a new method in the Class Explorer

This wizard inserts the definition of the method in the respective class and other specifications, e.g. for visibility or polymorphism, can be carried out here. If the method is not abstract (purely virtual), a function body is also generated in the source file.

Wizards are provided for inserting data elements and VCL properties.

DISPLAYING THE REFERENCES

The "References" tab of the Class Explorer displays all the source text positions for which the properties selected in the left area are used. Unfortunately, it is not possible to change on the spot from here on. Future versions will certainly see more enhancements for the new Class Explorer.

GRAPHIC DISPLAY OF CLASSES

A further tab shows you a graphic display. The class selected in the left area will be displayed on the "Graph" tab. You can also select several classes to show class hierarchies.

The following figure illustrates the class graph of an example application. Unfortunately, no usage links are displayed here, and partial inheritance is not recognized. I´m sure we´ll see a more in this regard in future versions.

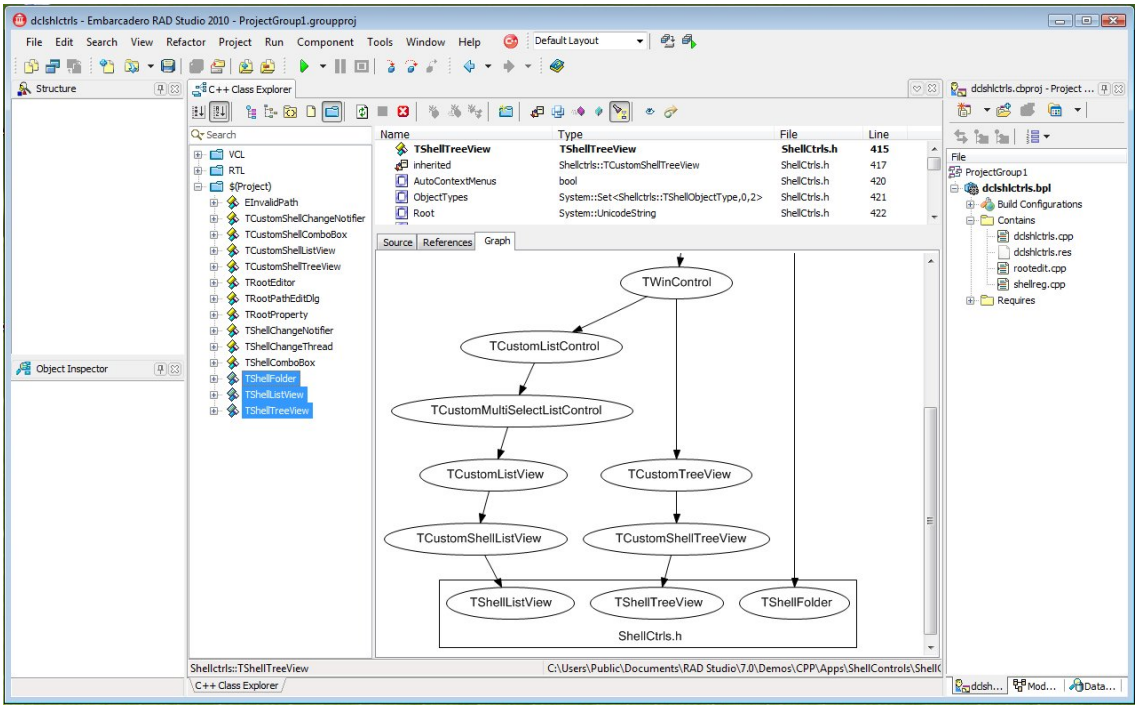

Figure 23: C++ Class Explorer in the graph view

TOOL PALETTE

The Tool Palette is dependent on the view selected in the workspace.

TOOL PALETTE IN DESIGN MODE

In design mode, the Tool Palette includes all the installed visual and non-visual VCL components. These are arranged in groups (categories, palettes). With a click of the mouse you can select the desired VCL components and place it on the currently edited form.

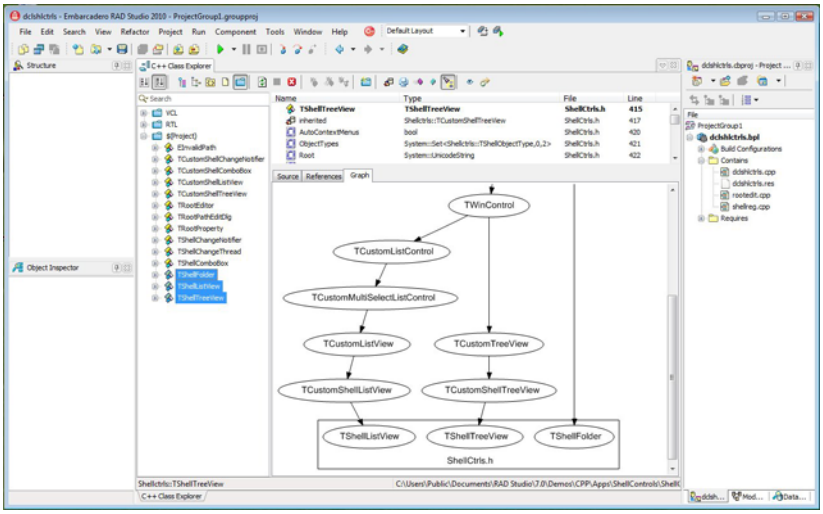

Figure 24: Tool Palette in design mode

One of the great advantages of the new Tool Palette is the quick search mode. If you enter the name (or part of this) in the input field in the upper part of the Tool Palette, then only those VCL components which meet this entry are displayed. This saves time and you don´t have to toggle between the various categories in order to find the VCL component you are looking for.

TOOL PALETTE IN THE CODE MODE

If you are working in the code window, the VCL components are hidden. The Tool Palette then includes an alternative view of the familiar New Items dialogue. From here you can create new projects or add to the existing project parts, for example, a form or a unit.

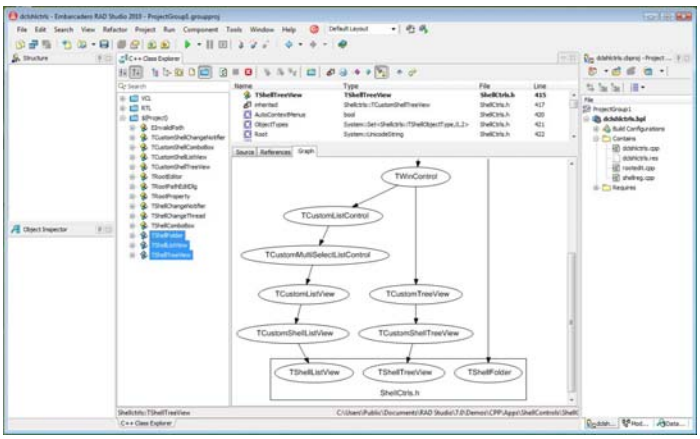

Figure 25: Tool Palette in the code mode

STRUCTURE VIEW

The information shown in the Structure view is dependent on the selection in the workspace.

STRUCTURE VIEW IN DESIGN MODE

If you work in the VCL designer, in Structure view all the VCL components which are located on the form are displayed hierarchically in a tree structure. All entries are displayed under the root (form). With that, the individual levels of the container are displayed and the controls are arranged within the container. Individual levels can be shown or hidden in the view.

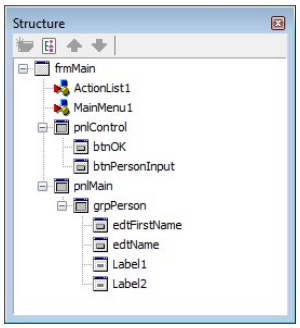

Figure 26: Structure view in design mode

If you click on an element in the Structure view it will be selected accordingly in the designer and in the Object Inspector, and vice versa. With drag & drop you can move the controls between the containers. Containers and controls can also be deleted, cut out or inserted in the view.

STRUCTURE VIEW IN THE CODE MODE

In the code mode, the Structure view refers to the structure of the source code which is open in the workspace. Therefore, this area is often referred to as the "Code Explorer".

If a header file is being edited in the code editor, you can see in the Structure view the classes, the properties and the methods that are defined in this header file. This gives you an overview, and, similar to the Class Explorer, the Structure view helps to navigate more quickly in the source text. By double clicking on a property or method you jump to the precise position in the source text.

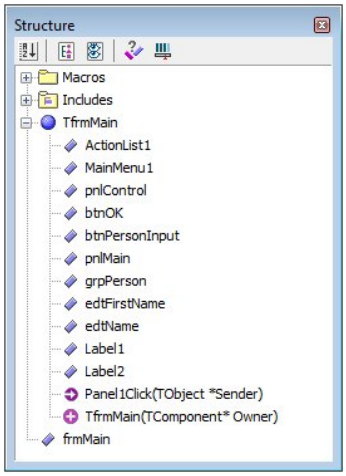

Figure 27: Structure view in the code mode

A source file displays the methods and variables that are being used in this file.

OBJECT INSPECTOR

The Object Inspector is used to check and change the properties and the behaviour of the selected form and the VCL components inserted in this. The Object Inspector area includes a selection box in which the currently selected VCL component is displayed. You use this selection box to select a different VCL component from the list. Alternatively, click on the desired VCL component directly in the designer or use the Structure view.

EDITING THE PROPERTIES OF THE VCL COMPONENTS

The published properties of the VCL components can be edited on the "Properties" tab. Here you can change the sorting and opt for a grouping based on content as opposed to an alphabetically sorted list.

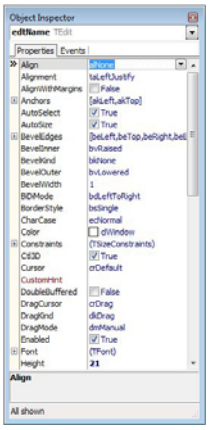

Figure 28: Object Inspector with properties

If the property is an enumeration, the values of the enumeration are shown in a selection box. In the case of yes / no values, an additional selection box appears in the value field. If the property is a VCL component, a [+] symbol precedes the name. You can expand this property and therefore also access the child properties.

If a property is a reference (pointer) to an instance of a different VCL component within the application, then the symbol [+] precedes this property and the name of the property is written in red. You can also expand this property and access the child properties of the other instance. To avoid any misunderstandings here, these names are also shown in colour (green). If an instance has not yet been assigned, a selection box showing all the suitable instances appears in the value field.

You double click on the field with the value of the property to search for the next possible value.

EDITING THE EVENTS OF THE VCL COMPONENTS

The handling methods for the events are set on the "Events" tab in the Object Inspector.

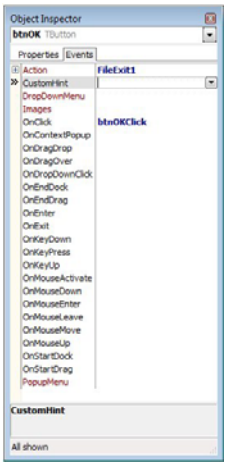

Figure 29: Object Inspector with events

In doing so, any methods which may already be available and are suitable are displayed in a selection box. To generate a new method, you double click with the left mouse button on the value field of the desired event. The development environment then writes the definition in the class and generates a suitable function body for this event handler. An automatic name is generated here. The new method is linked with the event in the Object Inspector and the source text with the function body is opened in the workspace.

If the name of the method is changed in the Object Inspector, this is also synchronized accordingly in the definition and implementation in the code.

FORM DESIGNER

The Form Designer is the core of RAD and the visual development takes place here. The windows of the application are generated and edited here in collaboration with the Tool Palette, the Structure view and the Object Inspector. As we already saw in the description of the Object Inspector, code fragments are generated and adopted in the code editor.

You can click on the VCL components that are provided in the Tool Palette and then place them on the form. The designer automatically generates identifiers which are made up of a designation for the respective VCL components and a consecutive number within the project (e.g. button1, button2, ...). In doing so, VCL components which are controls can be directly placed on the form or added to a container (e.g. a group or a panel). The mouse is used to align the controls or change their size, but there are also automatic methods available for arranging and aligning the elements within the form. All visual VCL components appear at runtime in the precise manner in which they are arranged in the design mode.

Elements can also be deleted, cut out, copied or pasted in the view.

The hierarchical structure of the container and controls on a form is shown in the Structure view and the properties of the controls are changed in the Object Inspector.

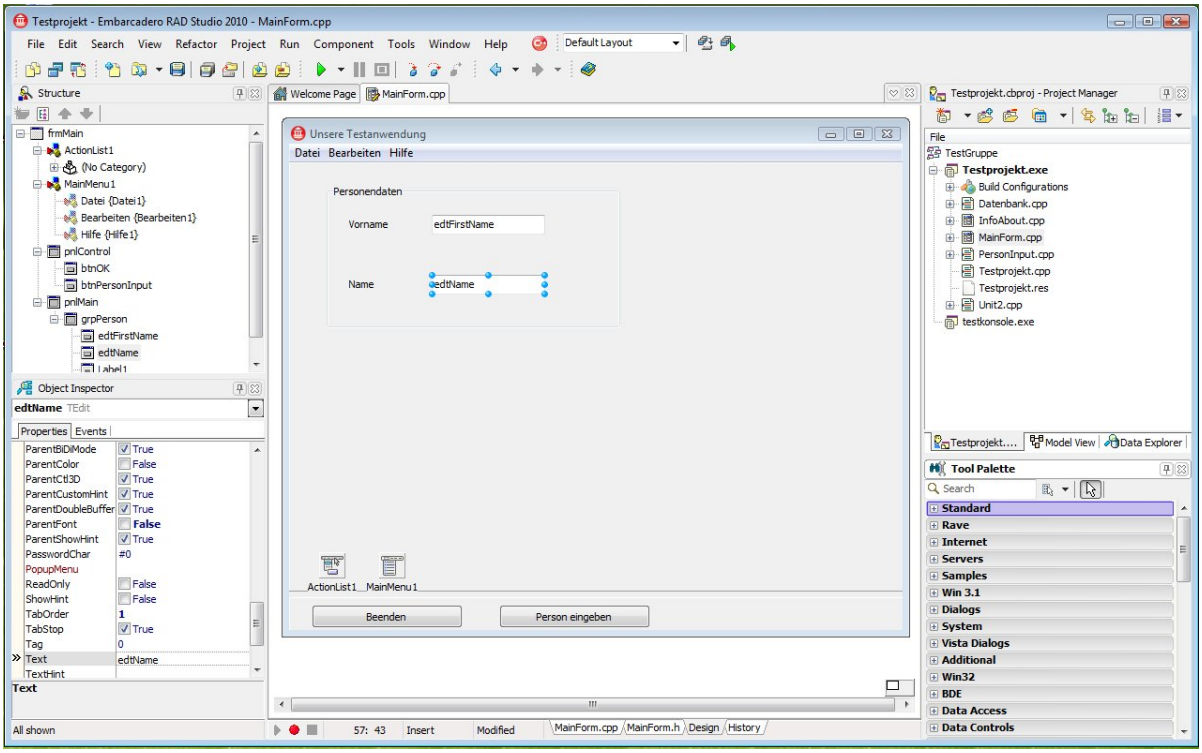

Figure 30: Form Designer

DEBUGGER

According to the current state of technology, it is not possible to write fault-free programs. While syntax errors can be found very quickly in program creation, the content errors are much more difficult to recognize. Modern development environments usually offer developers a special program (debugger) using which you can check the program flow and access important values which are otherwise not visible on the screen.

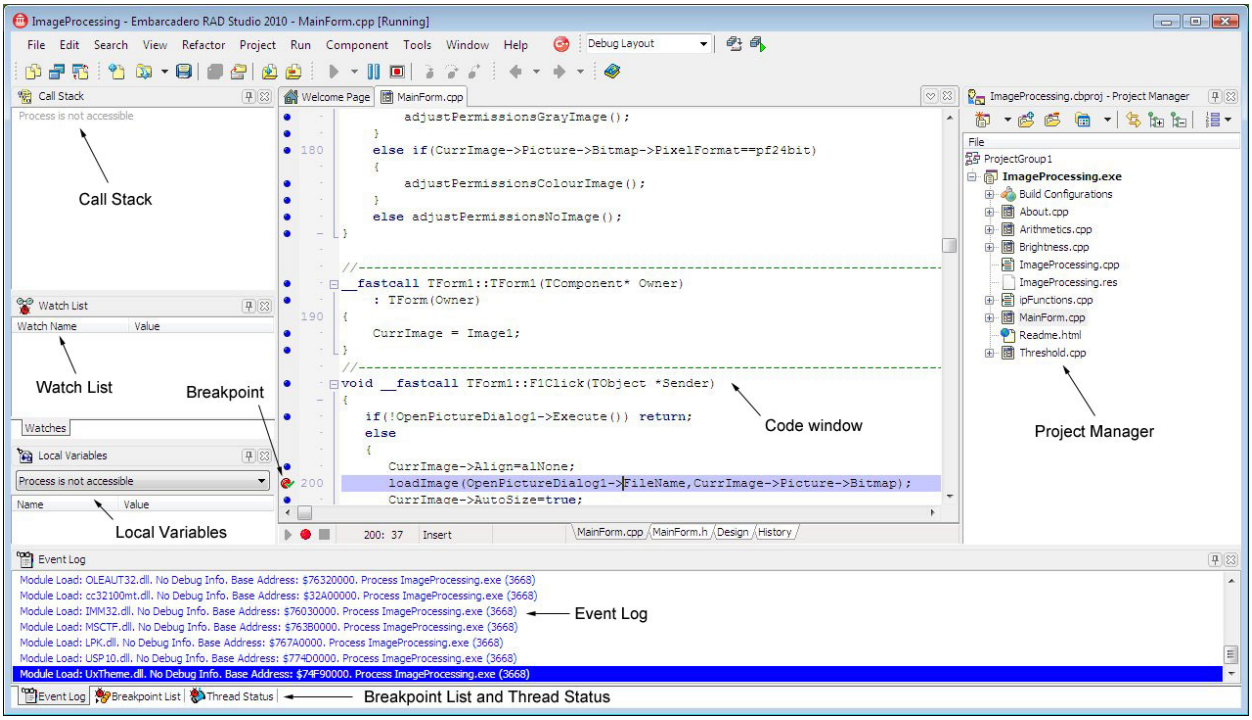

Figure 31: Development environment in debug layout

In C++Builder, the debugger is embedded in the development environment. When a program in the environment is launched, the debugger is automatically enabled, the view changes to the "Debug Layout" (you can also set up an independent debug layout).

A debugger requires special information to access the program information. This allows access to the variables, but also a synchronisation between the program flow and the respective position in the source text. If this information is missing, only the assembler code remains for debugging. If the program is compiled in debug mode, the compiler creates this information and the linker adopts it in the application.

WORKING WITH BREAKPOINTS

You can set breakpoints (at which the program execution will be interrupted) directly in the source text. To set a breakpoint, simply click on the border to the left in front of the desired position. The breakpoint is shown as a red point. To remove this, simply click on the point again. During the program execution, all possible positions at which the program can be stopped will appear in the code window denoted by a small blue point on the left border. Alternatively, you can also use the keyboard in order to set a breakpoint by pressing the F5 key when the cursor is at the desired position.

All the active breakpoints with additional information are displayed in the "Breakpoint List" view. Here you can delete or disable breakpoints or set some additional values.

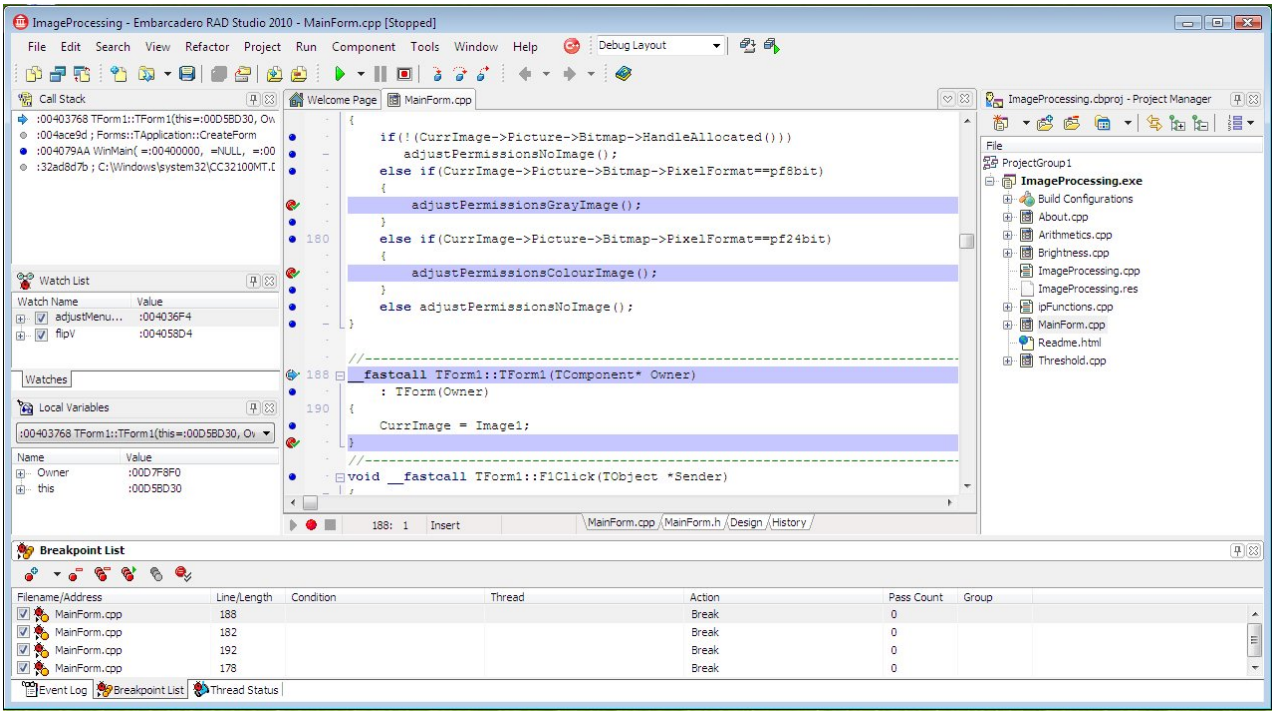

Figure 32: Debugger with the breakpoint list

FURTHER CONTROL OPTIONS AND VIEWS

You can choose from additional control options during debugging. You can execute the program up to the current source text position (Run to Cursor F4) or proceed through the program line by line. If you execute the program line by line, you can jump into the subroutines ((Trace Into F7)) or proceed with the same level (Step Over F8).

All the local variables which are currently located on the stack and their associated values are displayed in the "Local Variables" view. All the important variables that you wish to watch during debugging can be inserted in the "Watch List" view. To adopt a variable in the list of watched expressions, right click on this in the source text and then select the entry "Debug / Add watch at Cursor" from the context menu. Alternatively, you can also use the shortcut "Ctrl+F5".

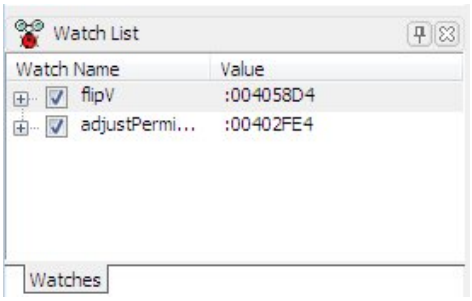

Figure 33: Debugger – Watch List

Via the "Debug / Evaluate / Modify" menu item or the shortcut "Ctrl+F7" a dialogue opens in which you can view the value of a variable and you can usually also change it.

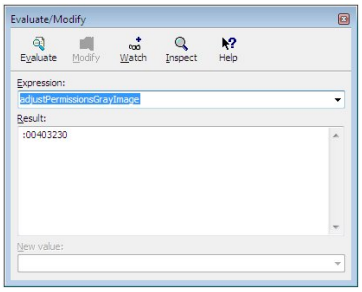

Figure 34: Debugger – evaluate and modify

Via the "Debug / Inspect" menu item of the context menu (Alt+F5) you launch the Debug Inspector with detailed information about the object. To obtain even more detailed information, you can expand this dialogue by simply double clicking with the left mouse button on the subentry.

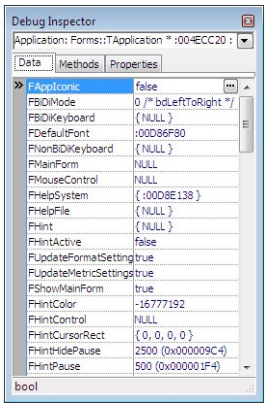

Figure 35: Debugger – Debug Inspector

In the "Call Stack" you can view the methods which were called until the breakpoint was reached.

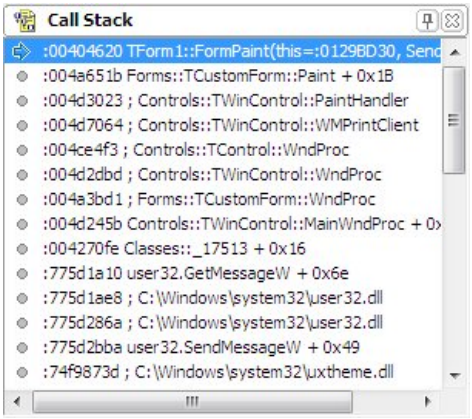

Figure 36: Debugger – Call Stack

By double clicking on a method in the call stack, the code view switches to the respective call position and you can view the parameters with which the method was called.

In the “Event Log” you can control the program flow and see which modules have been loaded and which processes were started.

NEW FEATURES IN THE BUILDER 2010

The previous versions had issues with displaying values for structured classes. A date would be displayed in the storage form of a floating point value, and a string as a structure. With C++Builder 2010 we see considerable improvements here and this data is now shown correctly. You can also add views for own classes by means of a special API.

Improvements have also been made in the area of threads. Now you can freeze individual threads or let them run independently of the others, therefore avoiding any interactions. Now, when debugging applications with threads, you can concentrate on what really matters.

If, “Tooltip expression evaluation” and “Tooltip symbol insight” are selected in the “Code Insight” options, in debug mode you simply hover the mouse over a variable in order to view the values and information directly as brief comments.

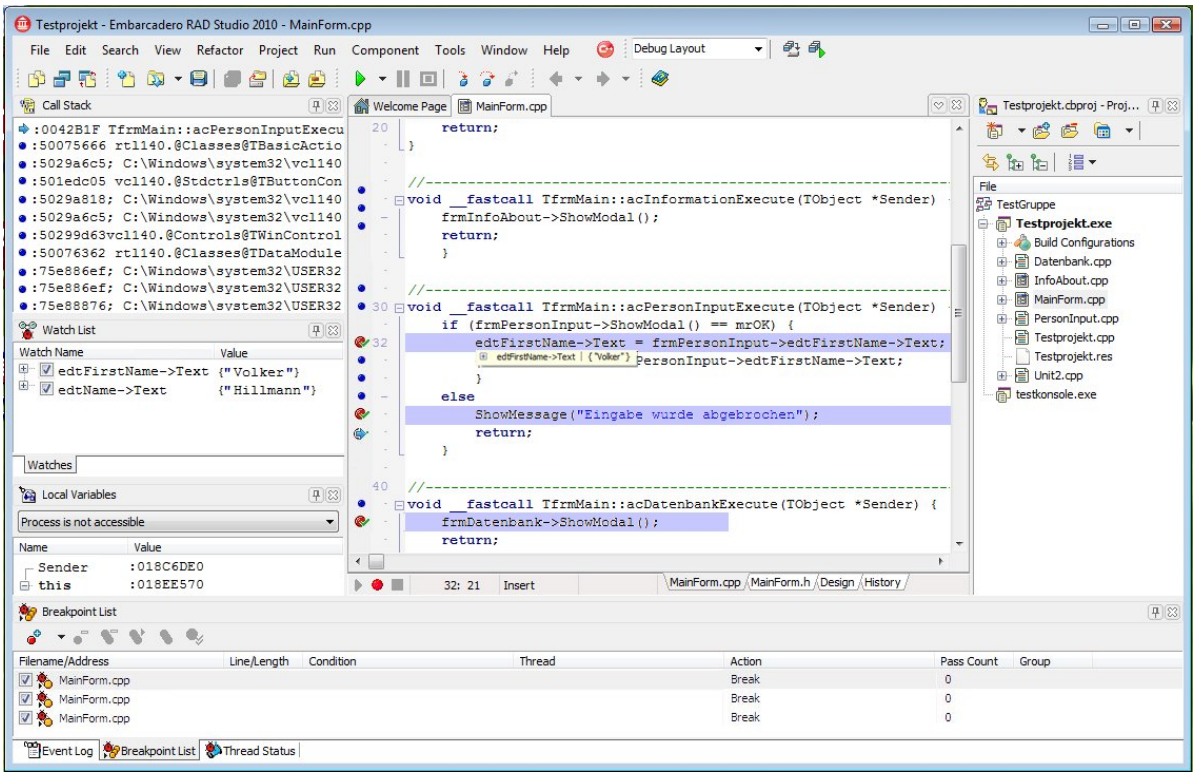

Figure 37: Debugger – Tooltip expression evaluation

You can use the structure information to access subordinate information. Additional information is recognised by the [+] symbol preceding the name of the variable.

ATTACHING TO A RUNNING PROCESS

Within the development environment you can also attach to a running process. If no debugger information is available and the source text is not accessible, only the assembler code of the process is displayed. In the debugger you can interrupt program execution at any time and free the memory.

IMPORTANT FILES

As the name already indicates, C++Builder is a development system for the programming language C/C++. But C++Builder also includes some other files, and the following table offers an overview of the key file types. Some file types have been adapted to the current versions of C++Builder so as to enable a clear separation from the older versions.

Name	Description
groupproj	Project group, incorporates several projects *(formerly *.bpg)*
cbproj	Project file, incorporates associated files *(formerly *.bpr)*
c	Source text file for C source file
cpp	Source text file for C++ source file
cxx	Alternative extension for C++ source text file
h	Header file for C/C++, definitions
hpp	Alternative extension for header file in C++
dfm	Interface descriptions in the Delphi format
rc	Interface descriptions in the Windows resources format
obj	File with compiled source file (machine code)
res	File with compiled resource (machine code)
lib	Static library (machine code) or imported library
dll	Dynamic link library (machine code)
exe	Executable file (machine code)
tds	File with symbol information for the debugger
pch	Pre-compiled header file *(formerly *.csm)*

Table 2: File types for C++Builder

CREATING A CONSOLE APPLICATION

The simplest way to start programming is to generate a console application. This is a text-based application for the Microsoft® Windows operating system, but it will run only on this operating system. Of course, text-based programs, which are restricted to the standard of C/C++, can be used on any other platform that has a C/C++ compiler. A compiler is then used to compile the sources on this platform.

The simplest method of generating a console application is to use the New Items dialogue and the console wizards. Here you can select the source type and some other settings. It is no longer possible to port any applications that use the VCL.

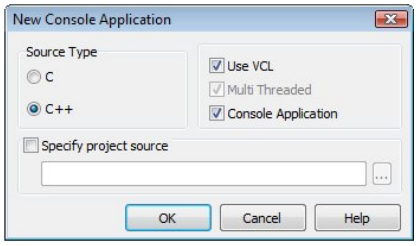

Figure 38: Wizard for the console application

The wizard uses the above settings to create the project file and the following source file with the main function of the program.

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include <tchar.h>

//---------------------------------------------------------------------------

#pragma argsused
int _tmain(int argc, _TCHAR* argv[]) {
   return 0;
   }
//---------------------------------------------------------------------------
```

Listing 3: Main function of the console application

Some would argue that the use of the header file <tchar.h> is new. This is usually adopted by most C/C++ compilers, especially on the Microsoft® Windows platform, so as to support the code transfer for international applications. The C standard extension in 1995 already intended for the C runtime library to be supplemented to include Unicode support. This involved the introduction of the integral data type wchar_t (wide character), which can represent any international character set. As C++Builder was also working with Unicode now, it was necessary to add this support for C/C++ programs.

The header file <tchar.h> makes available mappings for generic text for most data types. Therefore, the expression "_TCHAR" for compilers, which do not support Unicode, is translated into "char" and "_tmain" becomes the familiar method "main". Otherwise, the "_TCHAR" becomes the type wchar_t and the expression _tmain becomes the method "wmain".

VISUAL COMPONENT LIBRARY – THE RAD FRAMEWORK

RAD is the abbreviation for Rapid Application Development. Here, large parts of the program should be generated as quickly as possible in a visual environment without any direct programming. Predefined classes (VCL components) are made available for this purpose. RAD makes it possible to construct small and medium applications much more quickly. The VCL forms the basis of the RAD properties of C++Builder.

C++Builder has adopted the VCL from the Delphi development environment. This is why the object format of C++Builder had to be adapted accordingly. This also afforded a few syntax expansions, e.g. the key word "__published". From 1997 this did lead to a compartmentalisation in the C/C++ world, but also created the prerequisites for RAD and the use of various tools from the Delphi environment. The VCL was the requirement for both.

To make RAD possible, a new structure was introduced which was identified as a component within the VCL. Data and methods can be brought together as a property and possess default values. The programmer need only concentrate on any deviating property values. The properties can be published (visibility "__published") and then edited at design time in the Object Inspector.

Added to this are new, pre-defined methods, which are known as events. These form a program-specific connection to the messages. With that, the programmer must no longer deal directly with the complex event handling of the Microsoft® Windows operating systems. The main objective when designing the VCL was to hide the complex Windows API from the programmer.

All VCL components are derived via a simple inheritance structure from the basic class "TObject" and possess a property "Owner" in which the address of an instance is saved as an ancestor. In this way, all objects that are generated from the VCL components can be hierarchically linked. This means that the instances of the VCL components must always be generated dynamically so that a true ownership can be developed by the "Owner". When deleting an object, all the associated instances are recursively deleted.

There are visual and non-visual VCL components. While the visual components correspond to a concrete element on the screen (e.g. a menu, input field, button, …), the non-visual components provide extra services (e.g. a link to a database, a special selection dialogue,...). The visual VCL components possess a further property "Parent". This refers to the

superordinate container on the interface, so that the events of the operating system can be passed on and handled accordingly.

THE UNIT

The majority of C++ programmers don´t have much to say about this term, at least in connection with how it is used in C++Builder. The affinity of C++Builder with the proprietary development environment, Delphi, provides a satisfactory explanation for this. Pascal, the underlying programming language used here, stores the declarations and the implementations in one single file, they are just located in different areas. This file is referred to as a unit. This is different in C/C++, as declarations and implementations are separated. Nevertheless, this term has been adopted by C++Builder as a source file and the associated header file are combined to form a unit.

Nevertheless, C++Builder does not control this using the file name. The development environment uses an include guard for this. The following listing shows the header file of the empty unit with the name "TestUnit".

```
//---------------------------------------------------------------------------

#ifndef TestUnitH
#define TestUnitH
//---------------------------------------------------------------------------
#endif
```

Listing 4: Header file of an empty unit with the name "TestUnit"

The include guard corresponds to the name of the header file (however, without the point). This is mandatory in order for the development environment to link this file with the respective source file to a logical unit. If the name of the file and the include guard are not the same, there will be problems in the IDE. You can no longer toggle between the source and the header file, in design mode the forms are not displayed and search operations do not work.

The files that belong to a unit are displayed in a code window, in the lower area of the workspace you can toggle between the header and the source file (declaration) via the respective tabs. Each unit can house an unlimited number of declarations.

THE FORM

The term "form" is used in C++Builder in association with the window objects of Microsoft® Windows. In doing so, it doesn´t matter if this concerns a normal window or a dialogue. Each GUI application has at least one window, the so-called main window. Even if the form is not contained like the other VCL components in the Tool Palette, it is still a component. A form carries other VCL components, the controls which can be placed on this at a later date.

C++Builder creates a unit for each form. Moreover, there is another file in which the properties of the form class and those for the container and the controls that belong to this are saved. This

file has a proprietary format adopted from Delphi. The standard extension for these files is *.dfm.

As the form is a VCL component, the majority of properties can be edited in the Object Inspector, without any direct programming. When setting up a new project, the main form is generated alongside it.

You can also dock a form in a different container; this is then represented as part of this.

PROPERTIES OF A FORM

The following table contains an overview of the window properties which you can set in the Object Inspector at design time. There are default values for these properties, so you only have to worry about any deviating properties during the design.

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Action</td><td>Action that is assigned to this form. An action must be contained in a list, enabling a central management of responses to user inputs.</td></tr>
<tr><td>ActiveControl</td><td>Control on the form that currently has the focus.</td></tr>
<tr><td>Align</td><td>This property establishes the automatic alignment of the form to its parent container. In doing so, you can arrange several windows with the same property, one after the other. Possible values are:

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>alBottom</td><td>The form is aligned on the bottom border of the parent container.</td></tr>
<tr><td>alClient</td><td>The form assumes the entire area of the parent container.</td></tr>
<tr><td>alCustom</td><td>User-defined alignment of the form.</td></tr>
<tr><td>alLeft</td><td>The form is aligned on the left border of the parent container.</td></tr>
<tr><td>alNone</td><td>The window is not automatically aligned.</td></tr>
<tr><td>alRight</td><td>The form is aligned on the right border of the parent container.</td></tr>
<tr><td>alTop</td><td>The form is aligned on the top border of the parent container.</td></tr>
</table></td></tr>
<tr><td>AlignWithMargins</td><td>Attribute that establishes that the relative distance between controls on the form corresponds at least to the values set in the property “Margin”.</td></tr>
<tr><td>AlphaBlend</td><td>Attribute that establishes whether this form should be displayed transparently. AlphaBlending does not work on all systems.</td></tr>
<tr><td>AlphaBlendValue</td><td>Value between 0 and 255 which determines the degree of transparency for the AlphaBlending. The value 0 means that the form is completely transparent.</td></tr>
<tr><td>Anchors</td><td>Anchor points. These properties establish the position of the form after changes in size on the parent container.

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>akLeft</td><td>The left border of the window is anchored on the parent container.</td></tr>
<tr><td>akTop</td><td>The top border of the window is anchored on the parent container.</td></tr>
<tr><td>akRight</td><td>The right border of the window is anchored on the parent container.</td></tr>
<tr><td>akBottom</td><td>The bottom border of the window is anchored on the parent container.</td></tr>
</table></td></tr>
<tr><td>AutoScroll</td><td>Property that establishes that the scrollbars will appear automatically if required.</td></tr>
<tr><td>AutoSize</td><td>Attribute that establishes that the size of the form is automatically adapted to the content.</td></tr>
</table>

BIDIMode

Property that establishes the bidirectional mode. This determines whether a form can automatically adapt its appearance if the application is executed in a language environment in which text is read from right to left.

Value	Description
bdLeftToRight	Text is read from left to right. The alignment is not altered. The vertical scrollbar is shown on the right side of the control.
bdRightToLeft	Text is read from right to left. The alignment is altered. The vertical scrollbar is shown on the left side of the control.
bdRightToLeft NoAlign	Text is read from right to left. The alignment is not altered. The vertical scrollbar is shown on the left side of the control.
bdRightToLeft ReadingOnly	Text is read from right to left. The alignment and the scrollbar do not alter.

BorderIcons

This property determines which icons will be displayed in the title bar of a window. The property has the following values:

Value	Description
biSystemMenu	The system menu is displayed.
biMinimize	The minimize icon is displayed (icon).
biMaximize	The maximize icon is displayed (full screen).
biHelp	The help icon is displayed.

Specific combinations of this property are dependent on the property "BorderStyle".

BorderStyle

This property determines the nature and function of the window frame. (Attention: The size of windows can be changed only if they have borders). The type of margin is established by an enumeration.

Value	Description
bsDialog	The window is displayed as a dialogue. The size cannot be changed, the system menu and the icons for minimizing and maximizing are not displayed.
bsSingle	The size for this type of window cannot be changed, but the system menu and the icons for minimizing and maximizing are available.
bsNone	No visible frame, no change to the size.
bsSizeable	The window will be displayed with a standard frame, the size can be changed.
bsToolWindow	The window is displayed in the same manner as “bsSingle”, but the title bar is smaller.
bsSizeToolWin	The window is displayed in the same manner as “bsSizeable”, but the title bar is smaller.

BorderWidth

Property that establishes the border width of the window. Text and graphics which belong to child controls are shown within this border.

Caption

The form caption is shown in the title bar.

ClientHeight

Height of the client area in pixels.

ClientWidth

Width of the client area in pixels.

Color

Colour of the window (client area).

Constraints — Rules for the window (minimum size, maximum size)

Value	Description
MaxHeight	Maximum height of the window.
MaxWidth	Maximum width of the window.
MinHeight	Minimum height of the window.
MinWidth	Minimum width of the window.

Ctl3D — Attribute that establishes whether this form is displayed three-dimensionally.

Cursor — Type of mouse pointer in the client area of the window.

CustomHint — Component for user-defined display of hint texts in Vista style.

DefaultMonitor — In an environment with several monitors this property can establish on which monitor the form will be displayed.

Value	Description
dmDesktop	It is not attempted to display the window on a specific monitor.
dmPrimary	The window will be displayed on the monitor that is listed first in the "Monitors" property of the global screen object.
dmMainForm	The window is displayed on the same monitor as the main form of the application.
dmActiveForm	The window is displayed on the same monitor as the active form.

DockSite — Property that establishes whether other controls can be docked in this window.

DoubleBuffered — Property that establishes whether the content of a window is drawn directly (false) or written previously in a memory bitmap (true), which is then transferred.

DragKind — Attribute that establishes how the form will behave when it is dragged with the mouse (drag) (drag&drop or drag&dock).

Value	Description
dkDock	Automatic drag&dock function will be enabled at runtime.
dkDrag	Automatic drag&drop function will be enabled at runtime.

DragMode — Attribute that establishes when the form responds to the drag operation by the user.

Value	Description
dmAutomatic	Automatic drag&drop or drag&dock function will be enabled at runtime.
dmManual	Drag&drop or drag&dock function will be disabled at runtime.

Enabled — Attribute that establishes whether the window responds (true) or not (false) to the keyboard and mouse events.

Font — Font and attributes that should be used by the components in this window.

<table>
<tr><td>FormStyle</td><td>Type of window.

<table><tr><th>Value</th><th>Description</th></tr><tr><td>fsMDIChild</td><td>The window is a child MDI window.</td></tr><tr><td>fsMDIForm</td><td>The window is a parent MDI window.</td></tr><tr><td>fsNormal</td><td>The window is neither a parent nor a child MDI window.</td></tr><tr><td>fsStayOnTop</td><td>The window will always be displayed in the foreground, apart from if a different window with this property is being displayed over this. Attention: If a window with this property calls a further window of this type, none of the two will be displayed in the foreground.</td></tr></table></td></tr>
<tr><td>GlassFrame</td><td>Property that controls the compatibility with the Windows Vista Aero- System (glass effect). The properties of the TglassFrame component are

<table><tr><th>Value</th><th>Description</th></tr><tr><td>Enabled</td><td>Determines that the glass effect can be extended to the client area.</td></tr><tr><td>Left</td><td>Specifies how far (in pixels) the glass effect will be extended to the client area.</td></tr><tr><td>Top</td><td>Specifies how far (in pixels) the glass effect will be extended to the client area.</td></tr><tr><td>Right</td><td>Specifies how far (in pixels) the glass effect will be extended to the client area.</td></tr><tr><td>Bottom</td><td>Specifies how far (in pixels) the glass effect will be extended to the client area.</td></tr><tr><td>SheetOfGlass</td><td>If this property is set, the GlassFrame will be extended to the entire client area.</td></tr></table></td></tr>
<tr><td>Height</td><td>Total height of the window in pixels.</td></tr>
<tr><td>Hint</td><td>Hint text for the window – this text is displayed as a hint if the property “ShowHint” is set.</td></tr>
<tr><td>HorzScrollBar</td><td>For components which support horizontal scrollbars.</td></tr>
<tr><td>Icon</td><td>Icon for the window. Is shown minimized at the top left, otherwise at icon size if the window is minimized.</td></tr>
<tr><td>KeyPreview</td><td>Attribute that establishes whether keyboard events are passed first to the form and then to the controls (true) or first to the controls (false).</td></tr>
<tr><td>Left</td><td>Left position of the window on the screen (or relative to the parent container) in pixels.</td></tr>
<tr><td>Margins</td><td>Property that establishes the borders of the control. The possible values are:

<table><tr><th>Value</th><th>Description</th></tr><tr><td>Bottom</td><td>Bottom border of the window.</td></tr><tr><td>Left</td><td>Left border of the window.</td></tr><tr><td>Right</td><td>Right border of the window.</td></tr><tr><td>Top</td><td>Top border of the window.</td></tr></table></td></tr>
<tr><td>Menu</td><td>Name of the component for the main menu of the window.</td></tr>
<tr><td>Name</td><td>Name of the form (unique).</td></tr>
<tr><td>ObjectMenuItem</td><td>This property enables access to a menu of an OLE object, which can be enabled or disabled in accordance with the status of the object.</td></tr>
</table>

<table>
<tr><td>OldCreateOrder</td><td>This attribute controls whether the event OnCreate should be triggered after the constructor (false) or during the constructor and OnDestroy occurs before any destructors.</td></tr>
<tr><td>Padding</td><td>Component establishes the distance of the controls from the border.

<table><tr><th>Value</th><th>Description</th></tr><tr><td>Bottom</td><td>Distance from the bottom border of the window.</td></tr><tr><td>Left</td><td>Distance from the left border of the window.</td></tr><tr><td>Right</td><td>Distance from the right border of the window.</td></tr><tr><td>Top</td><td>Distance from the top border of the window.</td></tr></table></td></tr>
<tr><td>ParentBIDIMode</td><td>Attribute that establishes that the window adopts (true) or does not adopt (false) the BIDIMode attribute of the parent container.</td></tr>
<tr><td>ParentCustomHint</td><td>Attribute that establishes that the window adopts (true) or does not adopt (false) the property “ShowHint” of the parent container.</td></tr>
<tr><td>ParentFont</td><td>Attribute that establishes that the window uses (true) or does not use (false) the font of the parent container.</td></tr>
<tr><td>PixelsPerInch</td><td>Property that determines how the form is scaled in accordance with the current screen resolution. If this value is changed, it could be the case that the proportions are not maintained for all resolutions.</td></tr>
<tr><td>PopupMenu</td><td>Name of the component that represents the PopupMenu of the window.</td></tr>
<tr><td>PopupMode</td><td>This property establishes how the window behaves in terms of the Win32 style WS_POPUP. In the z-order, a window with the style is always above the owner. In connection with the property “PopupParent” you can avoid “hanging” applications.

<table><tr><th>Value</th><th>Description</th></tr><tr><td>pmAuto</td><td>Screen.ActiveForm used as with the “PopupParent” property.</td></tr><tr><td>pmExplicit</td><td>Direct assignment of a popup parent window. If this is not specified (zero), then Application.MainForm will implicitly be used as the PopupParent. If no Application.MainForm is assigned, then the Application.Handle will be used as the PopupParent.</td></tr><tr><td>pmNone</td><td>No control (behaviour corresponds to that of older versions).</td></tr></table></td></tr>
<tr><td>PopupParent</td><td>Basic window for popup control.</td></tr>
</table>

<table>
<tr><td>Position</td><td>Specifications for the dimensions and the position of the window.

<table><tr><th>Value</th><th>Description</th></tr><tr><td>poDesigned</td><td>The window is displayed at the same position and in the same dimensions as in the design.</td></tr><tr><td>poDefault</td><td>The dimensions and position of the window are established by the operating system.</td></tr><tr><td>poDefaultPosOnly</td><td>The window is displayed in the same dimensions as the design; the position is determined by the operating system.</td></tr><tr><td>poDefaultSizeOnly</td><td>The window is displayed at the position that was established at design time. The dimensions are determined by the operating system.</td></tr><tr><td>poScreenCenter</td><td>The window is displayed in the same dimensions as the design, but is displayed in the centre of the screen.
Attention: On systems with several monitors, you can move the window to a different monitor (property “DefaultMonitor”).</td></tr><tr><td>poDesktopCenter</td><td>The window is displayed in the same dimensions as the design, but is displayed in the centre of the screen.
Attention: No adaptation to systems with several monitors.</td></tr><tr><td>poMainFormCenter</td><td>The window is displayed in the same dimensions as the design, but is displayed in the centre of the main window.
Attention: No adaptation to systems with several monitors.</td></tr><tr><td>poOwnerFormCenter</td><td>The window is displayed in the same dimensions as the design, but is displayed in the centre of the window specified by the “Owner” property.</td></tr></table></td></tr>
<tr><td>PrintScale</td><td>Scaling the dimensions of the form for printout.

<table><tr><th>Value</th><th>Description</th></tr><tr><td>poNone</td><td>No scaling is executed for printout. This can cause stretching and compression.</td></tr><tr><td>poPrintToFit</td><td>The window is printed out at the proportions shown on the screen, but adapted to fit the size of the printed page.</td></tr><tr><td>poProportional</td><td>The window is printed out in such a way that it looks about the same on the screen and on the printout.</td></tr></table></td></tr>
<tr><td>Scaled</td><td>This property establishes whether a scaling should (true) or should not (false) be carried out in accordance with the difference in the font at design time and at runtime. If “Scaled” is true, the window changes its own and all child controls dimensions in order to maintain the relation between the dimensions of the controls and the height of the text displayed in the default font.</td></tr>
<tr><td>ScreenSnap</td><td>This property establishes whether the borders of the window are automatically aligned to the screen border if the user moves it. The distance required for this is determined in the property “SnapBuffer”.</td></tr>
<tr><td>ShowHint</td><td>Attribute that establishes whether (true) or not (false) the hint text should be displayed for the window.</td></tr>
<tr><td>SnapBuffer</td><td>This property establishes the maximum distance (in pixels) between the border of the window and the border of the screen, whereby no alignment has yet taken place. This is used in connection with the property “ScreenSnap".</td></tr>
<tr><td>Tag</td><td>User-defined value.</td></tr>
</table>

Top	Top position of the window on the screen (or relative to the parent container) in pixels.
Touch	Component for the touch and gesture control.
TransparentColor	Attribute that establishes that a colour in the form will be transparent. If the entire form should appear transparent, then the property "AlphaBlend" must be used.
TransparentColorValue	Colour of the window that should be displayed as transparent.
VertScrollBar	For components which support vertical scrollbars.
Visible	Attribute that establishes whether (true) or not (false) the window is visible.
Width	Width of the entire window in pixels.
WindowMenu	Use this property to access the window menu of the parent MDI window. This is available in all MDI applications.
WindowState	Value that establishes whether the window is displayed normal, maximised or minimised. Value / Description wsNormal – The window is displayed as normal. wsMinimized – The window is displayed as minimized. wsMaximized – The window is displayed as maximized.

Table 3: Properties of a form

EVENTS FOR A FORM

Messages play a key role in the Microsoft® Windows operating system. In C++Builder, the handling methods for events represent a link to these messages, and a set of events is predefined for each form. While some of the events correspond to direct messages, others are supplemented by the runtime system of C++Builder in such a way that programmers can better influence the process without too much effort.

The following table contains the events that are defined for a form.

Name	Description
OnActivate	Event is triggered when the window has the focus.
OnClick	Event is triggered when the user clicks with the left mouse button on the window.
OnClose	Event is triggered when the form is closed. The further handling is determined by the "Action" parameter. Value / Description caNone – The window may not be closed, there is no further processing. caHide – The window may not be closed, it is only hidden. The application can continue to access the window. (Default value for SDI windows) caFree – The window is closed and the memory freed. caMinimize – The window may not be closed, instead it is displayed as an icon. (Standard for unordered MDI windows)

Name	Description
OnCloseQuery	When exiting an application, the event "OnClose" is triggered only for the main form. Event is triggered before a window is closed. Within the event handling you can check if the window can be closed – the result is passed to the application by the "CanClose" parameter.
OnContextPopup	Event is triggered if the user opens the popup menu via the mouse or the keyboard (Shift+F10 or application key). (Windows message WM_CONTEXTMENU).
OnCreate	Event is triggered when the window is generated. Take note of the "OldCreateOrder" property.
OnDblClick	Event is triggered when the user double clicks using the left mouse button on the form.
OnDeactivate	Event is triggered when the form loses the focus. This event corresponds to the "OnActivate" event.
OnDestroy	Event is triggered when the window memory is freed. This event corresponds to the "OnCreate" event.
OnDockDrop	Event is triggered if a control is docked in this window. This event can only occur if the property "DockSite" is set to true.
OnDockOver	Event is triggered if a control is dragged over this window.
OnDragDrop	Event is triggered if a control is dropped on this window.
OnDragOver	Event is triggered if a control is dragged over this window.
OnEndDock	Event is triggered after dragging a control (either by dropping or cancelling).
OnGesture	Event is triggered when the user performs a gesture associated with this control.
OnGetSiteInfo	Event is triggered before an "OnDockOver" event is triggered, and returns the docking information to the control.
OnHelp	Event is triggered when the window receives a help request.
OnHide	Event is triggered when the window is hidden.
OnKeyDown	Event is triggered when the user presses a key while the window has the focus (note property "KeyPreview"). Windows message WM_KEYDOWN.
OnKeyPress	Event is triggered when the user presses a key. With this event, only characters of the ASCII character set are returned, hence it is not triggered in the case of special keys. Windows message WM_CHAR.
OnKeyUp	Event is triggered when the user releases a key.
OnMouseActivate	Event is triggered when the user presses a mouse button over the window, but the window does not have the focus.
OnMouseDown	Event is triggered when the user presses a mouse button over the window.
OnMouseEnter	Event is triggered while the mouse pointer enters the window.
OnMouseLeave	Event is triggered while the mouse pointer leaves the window.
OnMouseMove	Event is triggered while the mouse pointer moves over the window.
OnMouseUp	Event is triggered when the user releases a mouse button while the mouse pointer is positioned over the window.

Name	Description
OnMouseWheel	Event is triggered when the user turns the mouse wheel. If this event is not available, an event "OnMouseWheelDown" or "OnMouseWheelUp" is triggered depending on the direction of rotation.
OnMouseWheelDown	Event is triggered when the user turns the mouse wheel downwards.
OnMouseWheelUp	Event is triggered when the user turns the mouse wheel upwards.
OnPaint	Event is triggered when the window must be redrawn. The child controls are displayed only after this method has ended. Windows message WM_PAINT
OnResize	Event is triggered after the size of the window has changed.
OnShortCut	Event is triggered if the user presses a key in order to dispatch special keystrokes as ShortCut commands. Therefore, it is called prior to the event "OnKeyDown". If the keystroke is passed on, the parameter "Handled" must be set to true.
OnShow	Event is triggered when the window is made visible.
OnStartDock	Event is triggered when the user starts to drag the window and the property "DragKind" is set to the "dkDock" value.
OnUnDock	Event is triggered when the window is undocked.

Table 4: Events for the form

In addition to this, you can also set the properties of child components.

CONTROLS

Controls are visual components that correspond to the elements of the user interface. Some of these controls correspond to the Windows standard controls, while some are extensions or combinations of others.

As described in the Form Designer section, you select the respective control and place it on the form in the design view.

In C++Builder, these controls are divided into three separate groups:

- Standard
- Win32
- Additional

IMPORTANT CONTROLS IN THE "STANDARD" CATEGORY

This category contains the controls which are typical for all GUI environments. Independently of this grouping, applications which have been created with the VCL and C++Builder 2010 are not portable to another platform at this time.

Name	Description
Frames	Frames are containers for controls. They should support the reusability and can be reinserted in windows (or in other frames).

Name	Description
MainMenu	This non-visual VCL component is used to establish the main menu for a window. Having inserted this in a form, you then double click with the left mouse button on the component. This opens a wizard in which you can edit the structure of the menu.
PopupMenu	This non-visual VCL component is used to establish a popup menu (context menu) for a window, or an area of a window. This menu appears if the user presses the right mouse button or the respective Windows key. Having inserted this in a form, you then double click with the left mouse button on the component. This opens a wizard in which you can edit the structure of the menu.
Label	Control for displaying a text information in a window, e.g. as a description for other fields.
Edit	Control for displaying a standard input field in a window, suitable for adding or removing text information.
Memo	Control for displaying a multi-line input field in a window, particularly suitable for extensive, unformatted text information.
Button	Control for displaying a standard button in a window – can be extended using different properties.
CheckBox	Control for a check box in a window, which is either selected (clicked on) or not. Zero values are also possible.
RadioButton	Control for a radio button in a window. By using several radio buttons, the user can be presented with alternatives which are mutually exclusive. By default, all radio buttons in a parent container are summarised so that only one of the options can be selected at a time.
ListBox	Control for displaying a scrollable list in a window. The user can select an entry from the list and the position is returned via an index.
ComboBox	The control combines an entry field with a scrollable list. It is therefore classed as a combination field.
ScrollBar	Control for displaying a scrollbar with which you can move areas of a control or a window.
GroupBox	Control for displaying a group field in the window. This component is a parent container for other controls.
RadioGroup	Control for a group field that contains only radio buttons. The radio buttons must not be inserted manually; they are included in a list "Items" in the component. The selected value is set by the property "ItemIndex".
Panel	Control as an independent canvas (panel) in a window. This component is a parent container for other controls and can be emphasized three-dimensionally by a border. You can also use a panel as a surface for docking other containers or forms.
ActionList	This component is used to manage lists with actions. These can be assigned to controls, e.g. menus or buttons. The necessary properties are managed centrally by the action.

Table 5: Tool Palette – standard control elements

WIN32 CONTROLS

The "Win32" category contains the controls, which are implemented together with the Win32 API, and are typical for Windows programs.

Name	Description
TabControl	Control for several tabs in a window. The component is not made up of several pages which contain different controls, it is an independent object.
PageControl	Control for multiple-paged dialogue fields and tabs in a window. To insert a new page, click with the right mouse button and select the "New page" item at design time.
ImageList	This component is used to summarise and manage several icons or bitmaps. You access the individual graphics by means of an index (0..n-1).
RichEdit	This component is used to generate a standard RTF control which is used to edit formatted text.
TrackBar	Control with a track bar.
ProgressBar	Control with a progress bar.
UpDown	Control with up and down controller. This is made up of a pair of arrow buttons with which you can change numerical values.
HotKey	This component is used to establish shortcuts which enable direct access to actions and controls.
Animate	Control for displaying clips that consists of a sequence of image frames.
DateTimePicker	Control that is specifically used for entering date or time information. This is not a standard component, so there could be problems with the BIDI mode.
MonthCalendar	Control with a standard calendar in which you can set a date or respective date range.
TreeView	Control with a tree view, the nodes of which you can expand or collapse.
ListView	Control with a list. You can establish column headings and set various display modes.
HeaderControl	Control for table headers, the size of which you can change at runtime.
StatusBar	Control for an information area on the lower area of the window. This can consist of a text or several panels, one after the other.
ToolBar	Control with a container for buttons. All buttons have the same height and width. You can also add other controls.
CoolBar	Control for generating a Windows rebar control. Coolbars are containers for controls which can be moved and sized independently of each other. (Requires the files COMCTL32.DLL version 4.70 or higher).
PageScroller	Control with a special, scrollable display area. If this is larger than the parent container, arrows are added to the edges of the control to enable scrolling.
ComboBoxEx	Control with a combination field in which graphics and text indents

	can also be used in the drop down list. However, it is not possible to sort the list.
XPManifest	Component for supporting XP manifests in the application.
ShellRessources	This component is used for forward compatibility for Windows Vista system animations. As Vista programs are no longer allowed to access system resources, independent copies of these must be maintained. This component should be used only in programs that are intended to be executed under Vista (and higher).

Table 6: Tool Palette - Win32 control elements

ADDITIONAL

This category contains additional controls. It mainly concerns enhancements for existing controls to enable additional properties.

Name	Description
BitBtn	Control for bitmap buttons. These behave like normal buttons. Frequently used standard buttons can be depicted using the "Kind" property. Up to version C++Builder 2009, normal buttons can show only text.
SpeedButton	Control for individual buttons or groups which support graphics and different modes (unpressed, pressed, disabled). In earlier versions they were used to show a panel as a container toolbar.
MaskEdit	Control with a masked input form. The predefined masks are used to check the validity of the text entered by the user. The mask can also be used to format the entered text.
StringGrid	Control with a grid (table) of text fields. It provides properties and methods for working with a table.
DrawGrid	Control with a grid (table) of any data. The event handler "OnDrawCell" must be implemented in order to display this data.
Image	Control with a graphic element. This graphic could be an icon, a meta file or a different graphic file. Version 2009 and above also supports png graphics alongside bitmaps and jpeg graphics.
Shape	Control for simple geometric figures. You can establish outlines and fillings with this.
Bevel	Control for borders and lines with a bevel. This lends the element a three-dimensional effect that can appear either raised or lowered.
ScrollBox	Control for an area in a window in which those controls that extend beyond the thresholds of this area can be scrolled.
CheckListBox	Control with a list box in which the entries contain additional checkboxes which can be selected using the "Checked" property.
Splitter	Control for dividing the client area of a form into resizable panes.
StaticText	Control which behaves like a normal text field but that can be handled like a window.
LinkLabel	Control that contains HTML tags and links. The links appear as underlined text (as it is in a browser), and you click on the text to

Name	Description
	trigger the “OnLinkClick” event.
ControlBar	Control that controls the arrangement of components in a toolbar. This component serves to dock controls (usually toolbar elements).
ApplicationEvents	Help class that enables you to visually assign the events of the application. (Events can also be assigned in the source text without the class).
ValueListEditor	Control with a list of name / value pairs that can be edited.
LabeledEdit	Control with an input field and associated label. For this purpose, the input field is given the property “EditLabel” and you can control the position by means of the properties “LabelPosition” and “LabelSpacing”.
ButtonedEdit	Control with an input field that has two embedded buttons. These are optional and are referenced by the properties “LeftButton” and “RightButton”.
ColorBox	Control for a drop-down combo box in which the user can select colours. The colour selection is controlled by the “Style” property.
ColorListBox	Control for a scrollable list of all colours for which a constant is predefined.
CategoryButtons	Control for a group of buttons. These can be divided into categories and offer more flexibility than former toolbars. Buttons can be moved and copied.
ButtonGroup	Control for a container with associated buttons. You can control the display of the buttons with the property “ButtonOptions”.
DockTabSet	Control for a set of tabs which is similar to the divider pages of a notebook.
TabSet	Control for a set of horizontal tabs. The actions are triggered by clicking on the tab. The tab is automatically generated from the string list of the property "Tabs". (Provided only for backward compatibility).
TrayIcon	Control for an icon in the system tray next to the clock. This component provides methods for displaying the information in a balloon or for reacting to mouse clicks accordingly.
FlowPanel	Control for a container (panel) in which the child controls are displayed at predefined positions. This is controlled by the property “FlowStyle”.
GridPanel	Control for a container (panel) in which the controls can be positioned in a grid. When adding a new control, this is automatically inserted in the next free cell.
BalloonHint	Component for a control for displaying additional information (balloon style hints). The appearance is controlled by the “Styles” property. An instance of this component can be assigned to other controls in order to display the hint text accordingly. Hint text is separated using the “|” symbol. (Short text|Long text|Image ID) The hints are displayed in Vista in the themed style, in XP (themes enabled) in the Vista style or in XP (themes disabled) in the XP style.
CategoryPanelGroup	Control for a container with elements which can be hidden and shown. The elements are of type “CategoryPanel” and are added

Name	Description
	with the help of the context menu entry "New Panel".
ActionManager	This component is an auxiliary component, which provides methods for managing and displaying actions.
ActionMainMenuBar	This component renders action entries and displays these as menu items. It is a container; the respective entries are generated dynamically. The layout is automatically managed by an instance of "ActionManager".
PopupActionBar	This component extends the popup control by providing the option to associate actions with each menu item.
ActionToolBar	This component renders action entries and displays these as toolbars. It is a container; the respective entries are generated dynamically. The layout is automatically managed by an instance of "ActionManager".
XPColorMap	This component is used to establish the appearance of the action band components. The component contains a pre-defined assignment for the colour values. This achieves a uniform, coloured representation. Refer also to StandardColorMap and TwillightColorMap.
StandardColorMap	This component is used to establish the appearance of the action band components. The component contains a pre-defined assignment for the colour values. This achieves a uniform, coloured representation. Refer also to XPColorMap and TwillightColorMap.
TwillightColorMap	This component is used to establish the appearance of the action band components. The component contains a pre-defined assignment for the colour values. This achieves a uniform, coloured representation. Refer also to XPColorMap and StandardColorMap.
TCustomizeDlg	This component is used to generate a dialogue for customising the actions in the action bands.

Table 7: Tool Palette – additional controls

IMPORTANT PROPERTIES OF THE COMPONENTS

Each individual control has its own unique properties but also common ones. The following table contains the most important properties which are published by many controls.

Name	Description
Align	Alignment of the component within its parent component.
Anchors	Anchor points
Caption	Caption for the component.
Colour	Colour with which the component will be displayed.
Cursor	Shape of the mouse when it passes into the region covered by the control.
Enabled	Boolean value, component can be selected.
Font	Font
HelpContext	Integer with ID of the control for the context-sensitive help.
Height	Height of the component in pixels.
Hint	Hint text for this component.

Left	Left position of the component, relative to the parent container.
Name	(Delphi) name of the component (case sensitive, no file name).
Tag	Auxiliary property for the user-defined storing of information.
Top	Top position of the component, relative to the parent container.
Visible	Boolean value, whether or not component is visible.
Width	Integer with width of the component in pixels.

Table 8: Important properties of control elements

As for properties, there is a series of events which are published by many components. The following list contains the most important of these events.

IMPORTANT COMPONENT EVENTS

Name	Description
OnActivate	Is triggered when a component appears on the screen.
OnClick	Is triggered when the component is clicked on (left mouse button).
OnClose	Is triggered when the component is closed.
OnCreate	Is triggered when the component is created.
OnDeactivate	Is triggered when the component deactivated.
OnDestroy	Is triggered when the component is destroyed.
OnEnter	Is triggered when the component receives the focus.
OnExit	Is triggered when the component loses the focus.
OnHelp	Is triggered to provide help for the component.
OnHide	Is triggered when the component is hidden.
OnKeyPress	Is triggered when a key is pressed while the component has the focus.
OnMouseDown	Is triggered when a mouse button is clicked with the mouse pointer over the component.
OnMouseMove	Is triggered when the mouse is moved while the mouse pointer is over the component.
OnPaint	Is triggered in order to redraw the component.
OnShow	Is triggered when the component is displayed on the screen.

Table 9: Important events of controls

CREATING A VCL FORM APPLICATION

To create a GUI application for Windows, you select the "VCL Forms Application – C++Builder" item from the New Items dialogue. You open the New Items dialogue with the menu item "File / New / Other". You can also use the respective menu item in the File – New menu or in the Tool Palette. A wizard creates a new project file, a file with the source text for the project and the files for the main form. The main window opens in the design view.

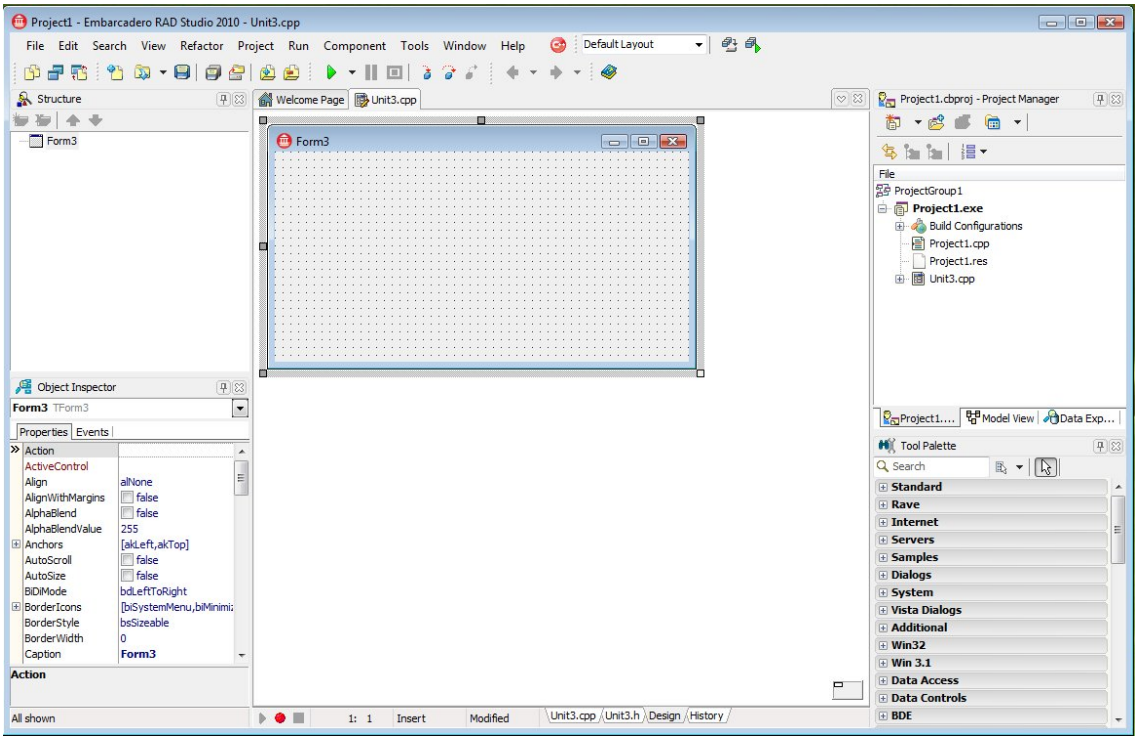

Figure 39: View of the IDE after creating a GUI project

Generally speaking, the files and components are created with a standard name and a consecutive number (Project1, Project2, … or Unit1, Unit2,…). The numbering always refers to the current directory. You can simply adopt these names and save the files accordingly. However, you should always allocate descriptive names to professional projects.

Firstly, you must always save the files of a project immediately, and, in this example, the name of the file with the main window is "MainForm.cpp" and the project is saved under the name "TestApp.cpp". In the Object Inspector you can change the properties of the main window, having first of all assigned a descriptive name for the main window. In this example, it is given the identifier "frmMain". The following table shows all the changes that have been made to the main window in the Object Inspector.

Components	Property	Value
frmMain	Caption	Test application for the C++Builder 2010
	Name	frmMain

Table 10: Settings for the main form

The following listing shows the project source text for the new application. A main form is automatically set up when you create a GUI application ("Application->CreateForm").

The application object contains important application data. Within the VCL framework, for this application object there is always an instance of the VCL component of type "TApplication" in a GUI application. This object is used to initialise the application, generate the windows and launch the event handling.

```
//---------------------------------------------------------------------------
// Test program
// adecc Systemhaus GmbH 2009
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <tchar.h>
//---------------------------------------------------------------------------
USEFORM("MainForm.cpp", frmMain);
//---------------------------------------------------------------------------
WINAPI _tWinMain(HINSTANCE, HINSTANCE, LPTSTR, int)
{
   try
   {
      Application->Initialize();
      Application->MainFormOnTaskBar = true;
      Application->CreateForm(__classid(TfrmMain), &frmMain);
      Application->Run();
   }
   catch (Exception &exception)
   {
      Application->ShowException(&exception);
   }
   catch (...)
   {
      try
      {
      throw Exception("");
      }
      catch (Exception &exception)
      {
         Application->ShowException(&exception);
      }
   }
   return 0;
}
```

Listing 5: Main program of the project

You can already launch this small application.

EXPANDING THE APPLICATION – WORKING WITH THE VCL COMPONENTS

The next step aims to expand the application. To do this, you add a second form in which entries can be made which should then be processed in the main form.

To add a form to the application, in the New Items dialogue you first select the area "C++Builder Files" from the tree view on the left side, and then the item "Form" from the entries on the right.

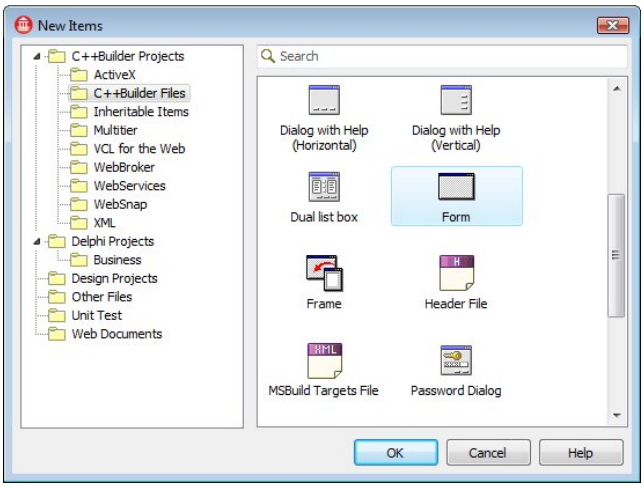

Figure 40: Creating a new form with the New Items dialogue

Alternatively, you can also generate a new form using the main menu. To do this, you select the menu item "File / New / Form - C++Builder". A form is then generated in the project and opened for editing in design mode. In this example we have saved the form with the name "DialogForm.cpp".

The new window is a dialogue and should appear in the centre of the application´s main window. You can edit this form accordingly in the Object Inspector. You can adjust the size of the form by means of the properties "Height" and "Width" or you can use the mouse.

Components	Property	Value
frmDialog	BorderStyle	bsDialog
	Caption	Input dialogue for the test application
	Name	frmDialog
	Position	poMainFormCenter

Table 11: Settings for the dialogue form of the main application

In the next step, you add the necessary VCL components to the form. This example aims to record personal data and an address. To do this, we position two groups (TGroupBox) on the form. These are located in the "Standard" tab of the Tool Palette. To add a VCL component, you left click on this in the Tool Palette and then left click at the position in the form where you wish to insert the VCL component. Four label controls ("TLabel"), two input fields ("TEdit"), one selection box ("TComboBox"), one control for inputting a date ("TDateTimePicker") are added to the first group and two buttons ("TButton") outside this group. The control for inputting a date is located in the "Win32" tab of the Tool Palette, while the others are located in the "Standard" tab.

We can align these controls accordingly at design time. There are several possibilities here. Firstly, you can of course align all the controls using the mouse (left click in the control, hold down and then drag the control) and you can also change the size (click a control, the control is highlighted with bullets, drag one of the bullets in the desired direction). If a group of controls has been previously selected (hold down the shift key, then click on the desired controls one after the other) you can then move these as a group. Alternatively, you can use the keyboard. After selecting a control, you can move it by simultaneously pressing the CTRL key and a cursor key. You can change the size with the shortcut "shift + cursor key". In both cases, guide lines are displayed which show the position in relation to the other VCL components.

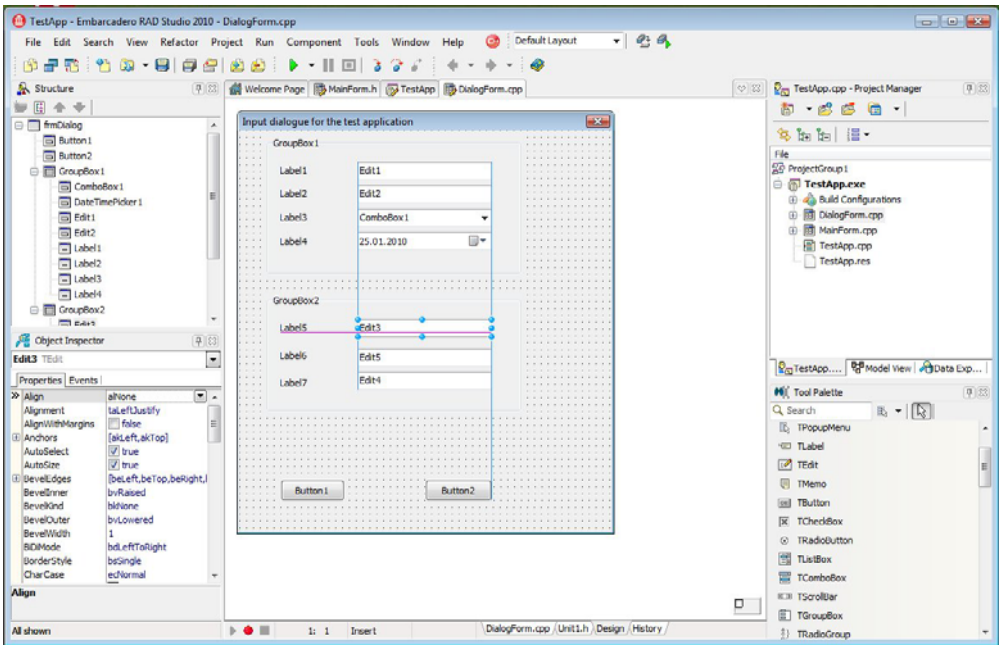

Figure 41: Working with VCL components, rough draft of the dialogue form

A third variant is the automatic alignment of groups. The menu item "Position / Align …" for this is located in the context menu. The following figures show the options available to you here.

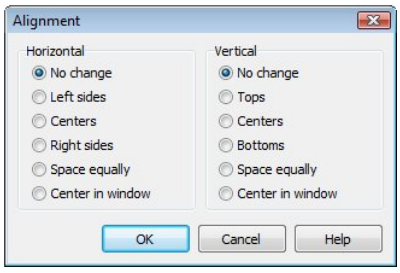

Figure 42: Wizard for aligning VCL components

The tab order that is used at runtime is very important when working with controls. In principle, the tab order of the controls follows the same order in which they were inserted in the form. Moreover, a property "TabOrder" (this can also be switched off using the property "TabStop") is assigned to each VCL component that is available with the TAB key. Nevertheless, it is difficult to constantly adapt these property values when adding (or moving) new controls. Therefore, there is a dialogue in which you can establish the order. This is located in the context menu of the design view under "Tab Order".

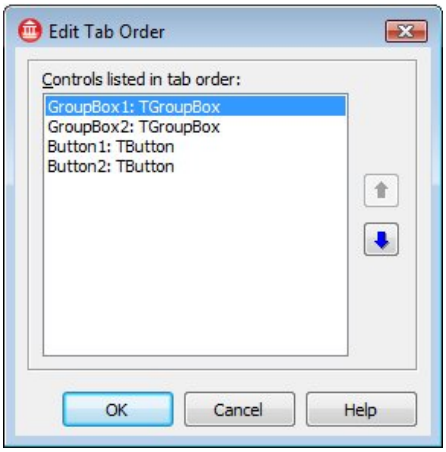

Figure 43: Dialogue for editing the tab order

In the next step, you edit the controls in the Object Inspector. In the following table, the descriptive names used in the application are listed in the left column and the default names as shown in Figure 41 are listed underneath in brackets.

Component	Property	Value
grpPerson (GroupBox1)	Caption	Personal data
	Hint	Person | Enter the personal data here.
	Name	grpPerson
	ShowHint	true
lblName (Label1)	Caption	Name
	FocusControl	edtName
	Name	lblName
edtName (Edit1)	Hint	Name | Enter the name of the person here.
	Name	edtName
	ShowHint	true
	Text	edtName
	TextHint	Name
lblFirstname (Label2)	Caption	First name
	FocusControl	edtFirstname
	Name	lblFirstname
edtFirstname (Edit2)	Hint	First name | Enter the first name of the person here.
	Name	edtFirstname
	ShowHint	true
	Text	edtFirstname
	TextHint	First name

Component	Property	Value
lblFormOfAddress (Label3)	Caption	Salutation
	FocusControl	cbxFormOfAddress
	Name	lblFormOfAddress
cbxFormOfAddress (ComboBox1)	Hint	Salutation | Select the salutation.
	Items	Mr Mrs
	Name	cbxFormOfAddress
	ShowHint	true
	Style	csDropDownList
	TextHint	Sex
lblBirthday (Label4)	Caption	Date of birth
	FocusControl	dtpBirthday
	Name	lblBirthday
dtpBirthday (DateTimePicker 1)	DateFormat	dfShort
	DateMode	dmComboBox
	Hint	Date of birth | Enter the date of birth here.
	Kind	dtkDate
	Name	dtpBirthday
	ShowHint	true
grpAddress (GroupBox2)	Caption	Address of the person
	Hint	Address | Enter the address of the person here.
	Name	grpAddress
	ShowHint	true
lblStreet (Label5)	Caption	Street
	FocusControl	edtStreet
	Name	lblStreet
edtStreet (Edit3)	Hint	Street | Enter the street here.
	Name	edtStreet
	ShowHint	true
	Text	edtStreet
	TextHint	Street
lblCity (Label6)	Caption	City
	FocusControl	edtCity
	Name	lblCity
edtCity (Edit4)	Hint	City | enter the city here.
	Name	edtCity
	ShowHint	true
	Text	edtCity
	TextHint	City
lblZipCode (Label7)	Caption	Postcode
	FocusControl	edtZipCode
	Name	lblZipCode
edtZipCode (Edit5)	Hint	Postcode | Enter the postcode here.
	Name	edtZipCode

Component	Property	Value
	ShowHint	true
	Text	edtZipCode
	TextHint	Postcode
btnOk (Button1)	Caption	Ok
	Default	true
	Hint	Confirm | End and accept data.
	ModulResult	mrOk
	Name	btnOk
	ShowHint	true
btnCancel (Button2)	Caption	Cancel
	Hint	Cancel | Close without accepting data.
	ModulResult	mrCancel
	Name	btnCancel
	ShowHint	true

Table 12: Working with VCL components, settings for the controls in the dialogue form

In the dialogue, the text fields show the names assigned in the Object Inspector. The controls should be initialized before they are displayed. The forms in the VCL have some events (refer to table 5 on page Error! Bookmark not defined.2), which are used here. You must use the event "OnCreate" to initialise the form. To insert this method, select the form and then in the Object Inspector under the "Events" tab you select the desired event "OnCreate". Then double left click on the value field. The method is then automatically inserted in the classes definition and as an empty function body in the implementation section. You can edit it in the code window. The following code shows the initialization of the controls.

```
void __fastcall TfrmDialog::FormCreate(TObject *Sender) {
   // Initialise data for the "Person" group
   edtName->Text               = "";
   edtFirstname->Text          = "";
   cbxFormOfAddress->ItemIndex = -1;
   dtpBirthday->Date           = TDateTime::CurrentDate();
   dtpBirthday->Time           = 0;

   // Initialise data for the "Address" group
   edtStreet->Text             = "";
   edtCity->Text               = "";
   edtZipCode->Text            = "";

   return;
   }
```

Listing 6: Working with the VCL, initialising a form

In principle, all forms that are added to a project will be generated automatically when the program starts and they then remain in the memory. If required, they are simply displayed and then closed again. In many cases this can be impractical, as the start phase is delayed and the memory requirement increases. You can change this in the project options. Select "Project / Options" and in the Options dialogue "Forms" on the left side. Here, with the help of the

buttons, you can move the forms between the two lists "Auto-create forms" and "Available forms". All the forms in the left list "Auto-create forms" are automatically generated when the program starts, whereas those in the right list "Available forms" must be generated independently as required. In addition to this, in the "Main form" combo box you can establish which form should be used as main window.

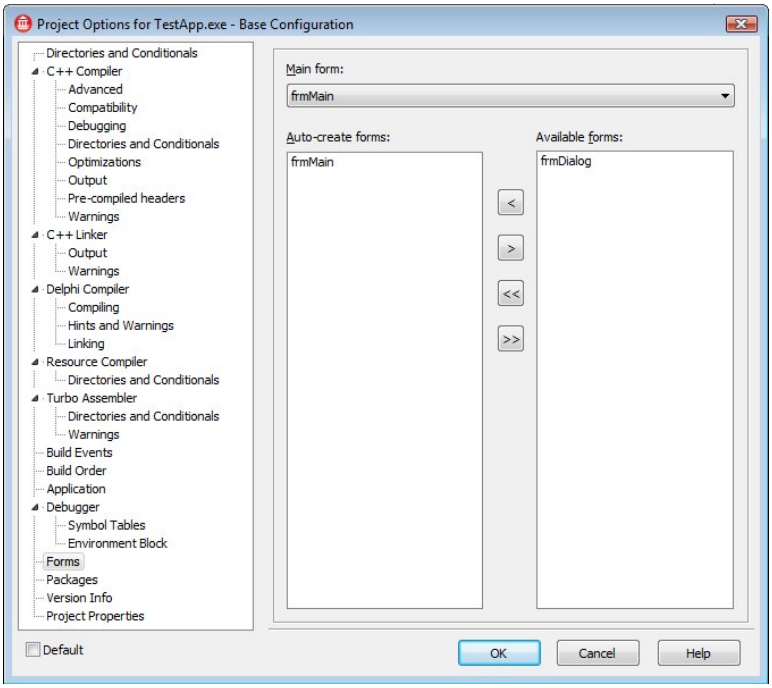

Figure 44: Working with VCL components, forms in the project options

In this example, the dialogue form will not be generated automatically.

We have thereby built into our application a simple input dialogue for personal data. Now, this dialogue must be called in the main program and the entries must be processed. For this purpose, four controls are added to the main program. To display the events for the input form, a multi-line text field is added ("TMemo") and also two buttons ("TButton"). The two buttons are arranged on a panel control ("TPanel"). The panel is positioned at the bottom border of the window and the text field uses the entire, remaining area.

The following table shows the necessary settings for the inserted controls. These will now be edited in the Object Inspector in accordance with the following specifications. Here too, the default names of the controls are shown in brackets for guidance purposes.

Component	Property	Value
pnlButtons (Panel1)	Align	alBottom
	Caption	*No caption (Delete the entry)*
	Name	pnlButtons
	ShowCaption	False
memResult (Memo1)	Align	alClient
	Lines	*No (Delete the entry in TStrings editor)*

Component	Property	Value
	Name	memResult
btnClose (Button1)	Caption	Close
	Default	true
	Hint	Close program | Close the program.
	Name	btnClose
	ShowHint	True
	Width	113
btnPerson (Button2)	Caption	Personal information
	Hint	Personal information | Enter the personal data.
	Name	btnPerson
	ShowHint	True
	Width	113

Table 13: Working with VCL components, properties of the controlsin the main form

Having set the properties, the main window should look like this.

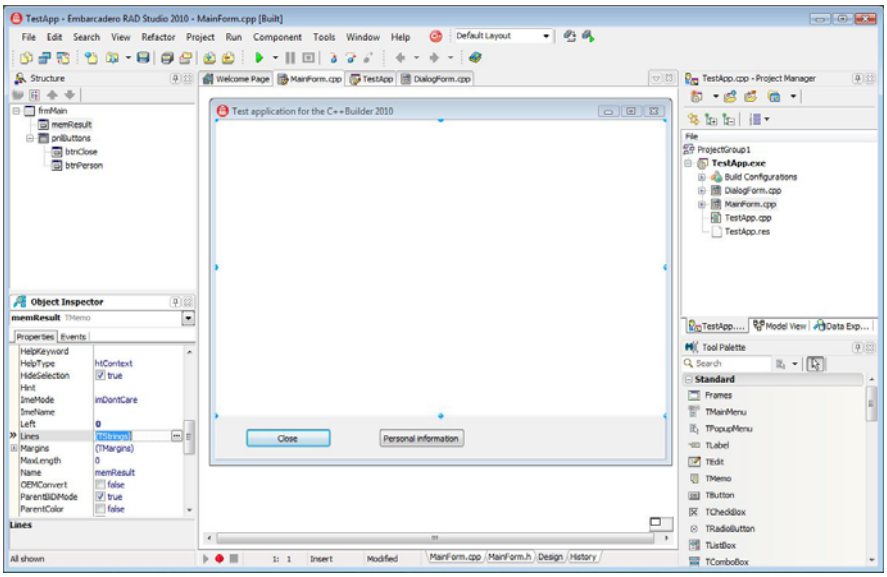

Figure 45: Working with VCL components, main form with components

You must now write the handling routines for the two button controls. Double click with the left mouse button on the button"btnClose", and the method "btnCloseClick" for the event "OnClick" is added to the classes definition and implementation section. The development environment switches to code mode in order for the method to be edited directly. However, in the Object Inspector you can select the "Events" tab, search for the desired event "OnClick" there and then double click on the value field.

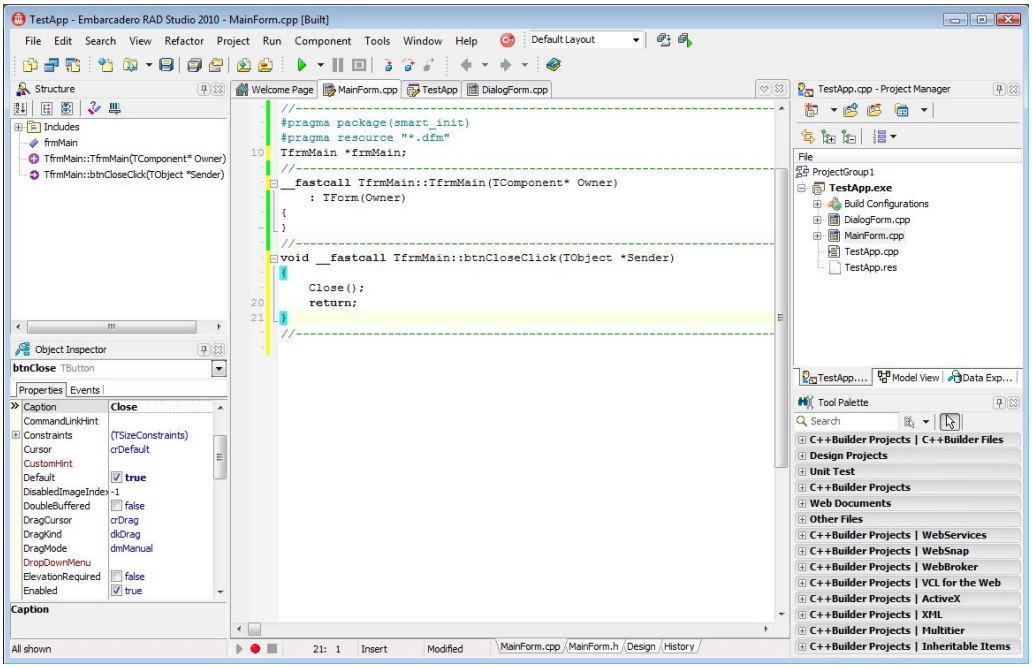

Figure 46: Working with VCL components, event handling

As no checks are performed in the program, you just call the method “Close()” in the main window in order to close the application.

Next, the input dialogue should be called and the data must be processed. You draw up the handling methods in the same manner as you did for the first button. To be able to use the personal dialogue in the main window, you must first of all add the definition of the dialogue to the implementation section of the header file “DialogForm.h”. C++Builder has a very useful function for this; via the main menu entry “File / Use unit” or the shortcut “Alt+F11” you can open a dialogue in which all the program parts that have not yet been used are displayed and can be added.

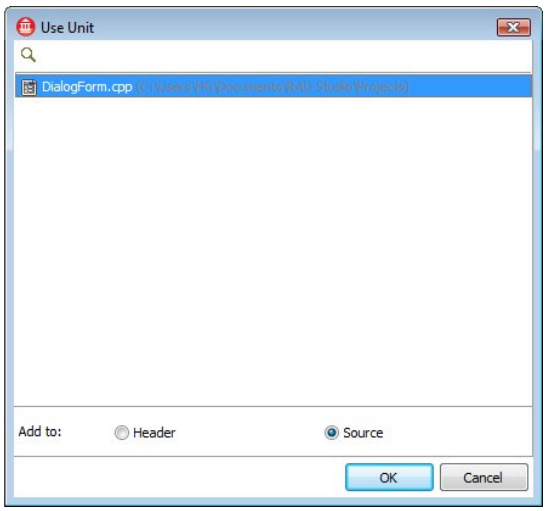

Figure 47: Working with the VCL, adding unused units

The following source text shows the complete file “MainForm.cpp”.

```
// ---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "MainForm.h"
#include "DialogForm.h"

#include <sstream>
#include <memory>
using namespace std;
// ---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmMain *frmMain;

// ---------------------------------------------------------------------------
__fastcall TfrmMain::TfrmMain(TComponent* Owner) : TForm(Owner) {
       }

// ---------------------------------------------------------------------------
void __fastcall TfrmMain::btnCloseClick(TObject *Sender) {
       Close();
       return;
       }

// ---------------------------------------------------------------------------
void __fastcall TfrmMain::btnPersonClick(TObject *Sender) {
   auto_ptr<TfrmDialog>dlg(new TfrmDialog(this));
   if (dlg->ShowModal() == mrOk) {
      wstringstream os;
      os << endl;
      if(dlg->cbxFormOfAddress->ItemIndex >= 0)
         os << dlg->cbxFormOfAddress->Text.c_str() << endl;

      if(dlg->edtFirstname->Text.Length() > 0)
         os << dlg->edtFirstname->Text.c_str() << " ";
      os << dlg->edtName->Text.c_str() << endl;

      os << dlg->edtStreet->Text.c_str() << endl;

      if(dlg->edtZipCode->Text.Length() > 0)
         os << dlg->edtZipCode->Text.c_str() << " ";
      os << dlg->edtCity->Text.c_str() << endl;

      os << endl
         << dlg->dtpBirthday->Date.DateTimeString().c_str() << endl
         << ends;
      memResult->Lines->Text = os.str().c_str();
      }
   else
      ShowMessage("Processing cancelled");
```

```
    return;
    }
// ---------------------------------------------------------------------------
```

Listing 7: Working with the VCL, complete source text of the main window

Here the data processing is limited to output a C++ stream which is then displayed in the text field "memResult", which was added to the main form. Access to the controls takes place directly. As the Delphi developers of the VCL have forgotten the necessary conversion functions for the C++ standard types, unfortunately we always need to fall back on the method "c_str()" in order to pass a VCL type to a C++ standard type.

The dialogue is generated dynamically and the address is saved in a smart pointer of type "auto_ptr". Smart pointers of this type have been part of the C++ standard since 1998. They ensure that, when ending the method (leaving the scope), the memory with the input form will be freed. In doing so, it is irrelevant whether the method ends properly or is cancelled after an error. Thus, since 1998 C++ has seen the safe avoidance of memory errors without a garbage collection, even if numerous comparisons with other commercial languages have actually left this impression. With the standard extension C++0X, we are seeing more smart pointers becoming standard.

FUNDAMENTAL COMMENT ON USING THE VCL

As a consultant in projects, I often find the comment in the source texts "Taken from the help". This is surprising, when you consider the magnitude and commercial significance of some of these projects. As far as the help or this paper is concerned, it is not a question of showing how the VCL components will be used in large-scale projects. The idea is to show how you can work with these components. This is why these examples can be used at most to create a prototype.

Generally speaking, you must recognise that the basic programming language of C++Builder is C/C++, and that all key parts must be written in this language. The VCL is merely a framework; and VCL types have no place in the business logic of an application, and access to the VCL components must be encapsulated accordingly. While the programming language C++ is standardized, almost 30% of all applications today are still created using C/C++, and development tools are offered by many manufacturers, the VCL framework and the programming language used here, Delphi, is owned by one organisation.

Irrespective of this, large-scale applications should always be structured in independent layers which are clearly separated from each other. This does not concern the use of proprietary technology, which the VCL also offers in the area "dbExpress" (formerly "Midas"), or the distribution across several computers, but the structuring of the application. In doing so, large-scale applications should have at least one data layer, one business layer and one presentation layer. It is also practical to have one independent layer with the classes which are used in the application and a further one for the processes. It is particularly the separation of the processes that is of importance today, as these are very dynamic and keywords like "SOA" are being bandied about.

UNICODE

Unicode is an international standard (ISO 10646) which should establish a long-term digital code for each meaningful character or text element from all known written cultures and character systems. The aim here is to eliminate the use of differing and incompatible encodings in the various different countries or cultural areas. After the first standard in the year 1991 which encoded only European, Near East and Indian characters, further versions followed in 1996, 1999, 2003, 2006 and 2008.

The Unicode is a multibyte character set (MBCS). Compared to earlier character sets, which mainly encoded only a specific font system, it is the objective of Unicode to encode all the font systems and characters that are in use. To achieve this, Unicode was initially defined as a 16-bit character set with a total of 65,536 characters. This code area quickly became insufficient, so, version 2.0 of Unicode saw an expansion to 17 planes each with a total of 65,536 characters, making available a total area of 1,114,112 characters. The current Unicode 5.1 allocates 100,713 code points to individual characters.

There are several variants. The following table contains a few examples:

Variant	Description
UTF-8	GNU, Linux, Unix, SMTP, WWW
UTF-16	Windows, OS X, Java, .Net
UTF-32	Simple encoding without variable byte length
UTF-EBCDIC	Derived from EBCDIC for IBM mainframes

Table 14: Variants of Unicode

Working with Unicode always begins in the operating system. To use respective character sets, these must be installed with a suitable keyboard layout. In the operating systems Microsoft® Windows Vista and Microsoft® Windows 7, the language control is located in system tray of the task bar.

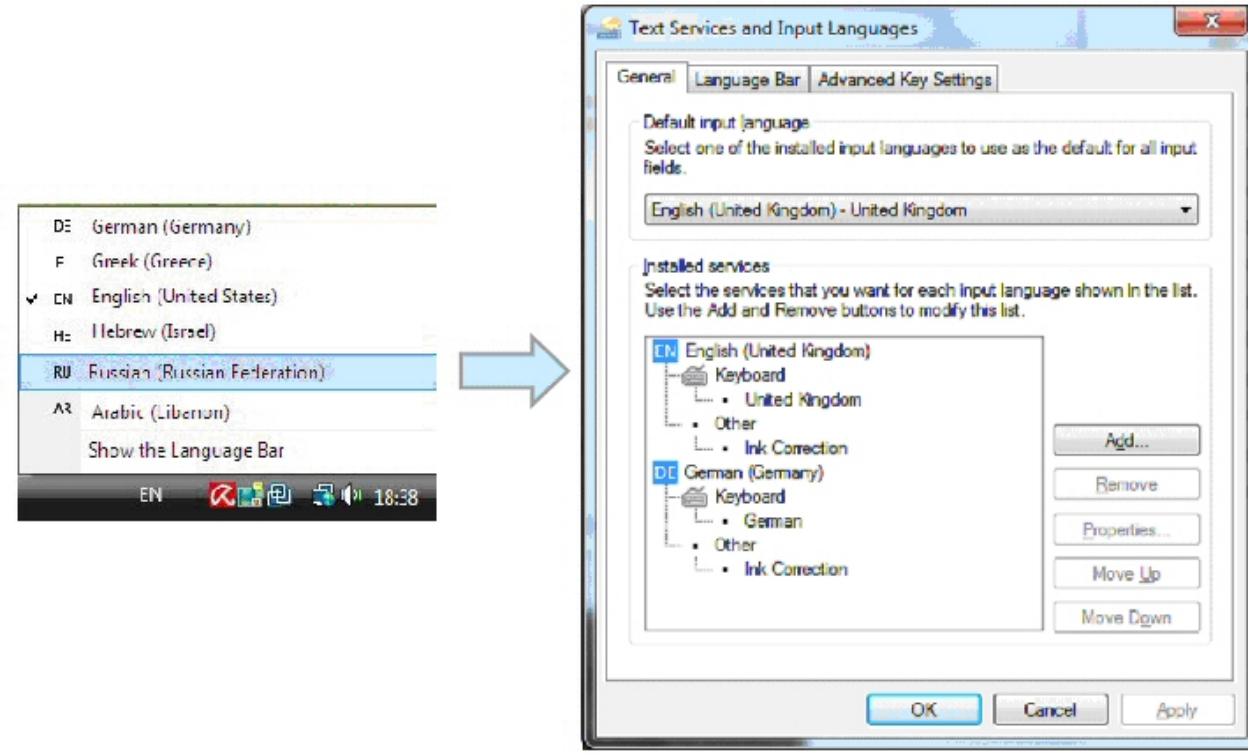

Figure 48: Unicode selection in the operating system

The new data type "UnicodeString" is inserted in the VCL in order to support the appropriate character set. This, however, should not be confused with the data type "WideString", which corresponds to the Windows-specific data type "BSTR" which is used for the COM automation.

Up until now, the data type "String" used in the VCL framework (and in the majority of programs which have been written using C++Builder) was a replacement for the VCL data type "AnsiString" and was used as a character string data type in all VCL components. Since version C++Builder 2009, the "String" type has become a pseudonym for the new data type "UnicodeString".

The C99 standard already introduced the data type "wchar_t" for displaying any country-specific character set in the programming language C. This type was adopted accordingly in the C++ standard, and the associated string type in the STL is "wstring". This is located in the header file "string". Other types in C/C++ are "char16_t" for characters in the UTF-16 character set and "char32_t" for the UTF-32 character set. As C++ uses a strict typification, it can cause problems when changing to a current version of the C++Builder.

The VCL data type "UnicodeString" also internally uses the "wchar_t" character type, and so the method "c_str()" of the class now also returns a pointer to this type.

UNICODE IN THE VCL COMPONENTS

The following figure shows the use of Unicode in the VCL components.

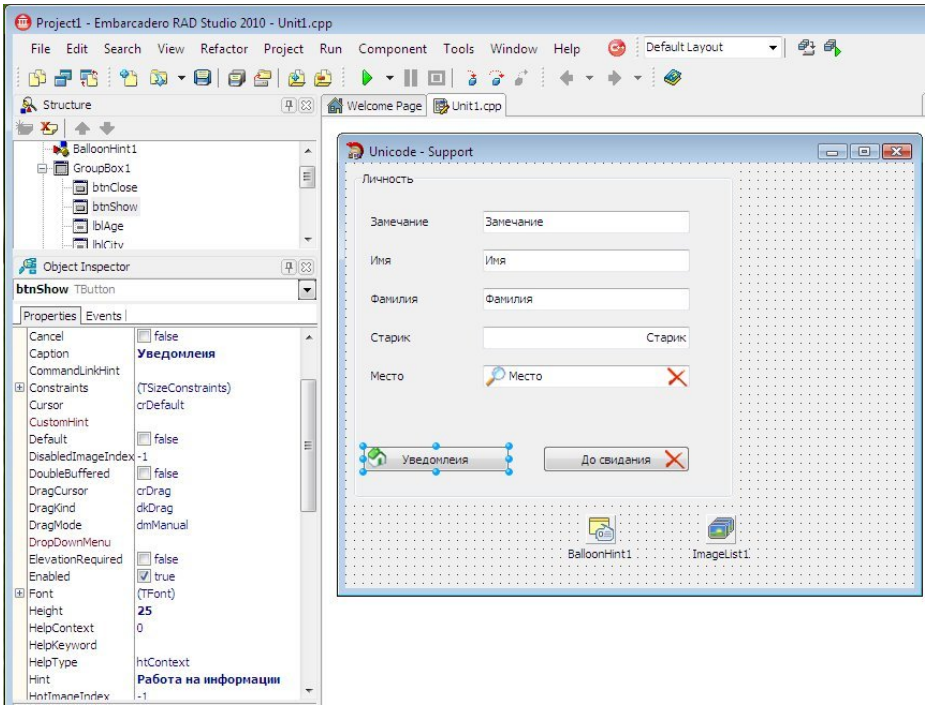

Figure 49: Unicode in the VCL components

The component "btnShow" has been selected in the figure. The properties "Caption" and "Hint" are character strings and these can be entered once you have switched to an international character set. Thanks to the synchronisation between the Object Inspector and the Form Designer, you can see immediate results. This enables Unicode to be seamlessly embedded in the VCL framework and the RAD environment.

UNICODE IN THE SOURCE TEXT

Naturally, you can also use Unicode directly in the source text. However, there are some points to consider.

STRING LITERALS FOR UNICODE CONSTANTS IN THE SOURCE TEXT

The following code shows a dialogue in Russian, in which a list will be filled according to a selection in a listbox. As the use of the character sets is kept completely separate in C/C++, when entering in the source you must specify a string literal, as the compiler will otherwise use a national encoding. Common literals are "u" for UTF-16 and "U" for UTF-32. The literal "u8" for UTF-8 is not supported. The literal used so far "L" for "wchar_t" encoding remains valid.

```
#include <utility>
#include <memory>
#include <string>
#include <list>
using namespace std;

#include "OrtsauswahlForm.h"

void __fastcall TfrmUnicode::edtOrtLeftButtonClick(TObject *Sender) {
   typedef list<pair<String, String> >    Orte;
   typedef Orte::iterator                 itOrte;
   typedef Orte::value_type               valOrte;

   #pragma region Initialisierung
   Orte theOrte;
   theOrte.push_back(valOrte(u"Берлин",    u"Berlin"));    // Berlin : Deutschland
   theOrte.push_back(valOrte(u"Хельсинки", u"hɛlsiŋki")); // Helsinki : Finnland
   theOrte.push_back(valOrte(u"Бухарест",  u"Bucureşti")); // Bukarest : Rumänien
   theOrte.push_back(valOrte(u"Москва",    u"Москва"));    // Moskau : Russland
   theOrte.push_back(valOrte(u"Рига",      u"Rīga"));      // Riga : Lettland
   theOrte.push_back(valOrte(u"Минск",     u"Мінск"));     // Minsk : Weißrussland
   theOrte.push_back(valOrte(u"Афины",     u"Αθήνα"));     // Athen : Griechenland
   theOrte.push_back(valOrte(u"Никосия",   u"Λευκωσία"));  // Nikosia : Zypern
   theOrte.push_back(valOrte(u"Стамбул",   u"İstanbul"));  // Istanbul . Türkei
   theOrte.push_back(valOrte(u"Бейрут",    u"بيروت"));     // Beirut : Libanon
   theOrte.push_back(valOrte(u"Амман",     u"عمان"));      // Amman : Jordanien
   theOrte.push_back(valOrte(u"Дамаск",    u"دمشق"));      // Damaskus : Syrien
   theOrte.push_back(valOrte(u"Эр-Рияд",   u"الرياض"));    // Riad : Saudi-Arabien
   theOrte.push_back(valOrte(u"Иерусалим", u"ירושלים"));   // Jerusalem : Israel
   #pragma end_region

   auto_ptr<TfrmOrtsauswahl> frm (new TfrmOrtsauswahl(this));
   for(itOrte it = theOrte.begin(); it != theOrte.end(); it++) {
      TListItem* item = frm->ListView1->Items->Add();
      item->Caption = it->first;
      item->SubItems->Add(it->second);
      }

   TListColumn  *NewColumn;
   frm->ListView1->Columns->Clear();
   NewColumn = frm->ListView1->Columns->Add();
   NewColumn->Alignment = taLeftJustify;
   NewColumn->Caption   = u"место";
   NewColumn->Width     = 200;

   NewColumn = frm->ListView1->Columns->Add();
   NewColumn->Alignment    = taLeftJustify;
   NewColumn->Caption      = u"Место в языке страны";
   NewColumn->Width        = 300;

   frm->Caption            = u"Выберите место";
   frm->btnOk->Caption     = u"Принимать";
   frm->btnCancel->Caption = u"Прекращение";
```

```
if(frm->ShowModal() == mrOk) {
   TListItem* item = frm->ListView1->Selected;
   if(item != 0) {
      edtOrt->Text = item->SubItems->Strings[0];
      }
   }
return;
}
```

Listing 8: Unicode in the source text

UNICODE IN THE INPUT AND OUTPUT STREAM

Input and output streams play a significant part in the programming language. The appropriate operators are implemented for the standard data types, and there is of course a tremendous advantage in the fact that no additional data conversions (e.g. integer to string with the VCL method "IntToStr") are required for the input and output. As Unicode and the international character type "wchar_t" have been standard in C/C++ for over 10 years now, input and output streams can be used.

The string streams are a special feature, in that character buffers can be used for the input and output. Historically, the class "strstream" has been used (additional variants for input or output only are "istrstream" and "ostrstream"), which is defined in the header file "strstream". This class is described in most C++ books. However, the buffer here is based on the national character type "char" and it is therefore not possible to use it for Unicode.

Nevertheless, this type and its associated header file were changed with the 1998 standardisation. The class "strstream" is only available in C++ for compatibility reasons, but was not extended any further. Unfortunately, many C++ books still use the old classes. The new input and output streams are based on templates as a constituent of the STL.

The header file that should now be used in C++ programs is "sstream", the respective class "stringstream" (also here, with "istringstream" and "ostringstream", there are appropriate extensions which are specialized only for input or output). The national character set is used in these classes and the main difference lies in the method "stringstream::str()", which returns a value of type "string" (the method "strstream::str()" returns a pointer to a character of type '"char").

The header file "sstream" also has implementations for use with international character sets. The respective type in the STL is "wstringstream" (additionally "wistringstream" and "wostringstream"). Here the method "wstringstream::str()" returns a value of type "wstring".

For the following dialogue a handling routine is needed in order to output the data via a stream that has been entered.

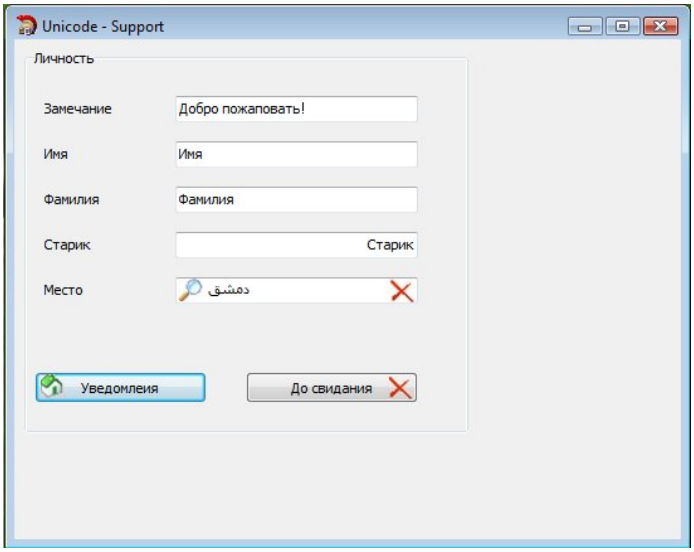

Figure 50: VCL dialogue with Unicode

```
#include <sstream>
using namespace std;

void __fastcall TfrmUnicode::btnShowClick(TObject *Sender) {
   wstringstream os;
   os << edtBemerkung->Text.c_str() << endl
      << edtOrt->Text.c_str() << ends;

   ShowMessage(os.str().c_str());
   return;
   }
```

Listing 9: Use of "wstringstream"

The source code above results in output shown in figure 50. As the method "wstrstream::str()" returns a value of type "wstring", and no conversion operators are available for the STL standard types in the VCL, the method "wstring::c_str()" must still be called and this returns a pointer to the data type "wchar_t".

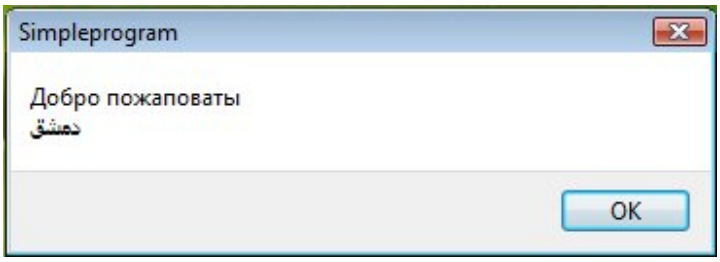

Figure 51: Use of "wstringstream", output of the handling method

CONVERTING UNICODE IN THE NATIONAL CHARACTER SET

In C++ programs it can be necessary to convert the Unicode to the former national character set. The following method shows how you can realise this in the C/C++ standardised methods.

```
#include <memory>
#include <cstdlib>

using namespace std;

int Konvertiere(std::string& target, String const& source) {
   size_t origsize  = wcslen(source.w_str());
   size_t converted = 0;
   auto_ptr<char> ptrTarget(new char[origsize + 1]);
   do {
      size_t curconverted = wcstombs(ptrTarget.get() + converted,
                                     source.w_str() + converted, origsize);
      if(curconverted == -1) curconverted = strlen(ptrTarget.get() ) - converted;
      converted += curconverted;
      if( converted < origsize ) {
         ptrTarget.get()[converted] = '?';
         converted += 1;
         }
      }
   while(converted < origsize);
   ptrTarget.get()[origsize] = 0;
   target = ptrTarget.get();
   return (int)converted;
   }
```

Listing 10: Converting Unicode in a string

In the source text example above, all the characters which cannot be converted to the national character set are converted into a "?". This is then followed by the conversion. In critical applications, it can make sense to trigger an exception in this case and handle this as an error.

There are automatic type conversions for data types "AnsiString" and "UnicodeString". For this purpose, the handling method implemented in Listing 9: Use of "wstringstream" will be changed, but first the contents of the input fields will be converted to the type "AnsiString".

```
#include <sstream>
using namespace std;

void __fastcall TfrmUnicode::btnShowClick(TObject *Sender) {
   AnsiString strBemerkung = edtBemerkung->Text;
   AnsiString strOrt       = edtOrt->Text;
   wstringstream os;
   os << strBemerkung.c_str() << endl
      << strOrt.c_str() << ends;

   ShowMessage(os.str().c_str());
   return;
   }
```

Listing 11: Automatic conversion of "UnicodeString" to "AnsiString"

However, then all those characters that cannot be mapped to the national character set are lost (without any warning). This results in the following output.

Figure 52: Output of Unicode after the automatic conversion to "AnsiString"

Therefore, in a productive application, it is always preferable to use a controlled conversion.

CREATING DATABASE APPLICATIONS

One of the advantages of Embarcadero C++Builder is integrating corporate data in the applications. This is where the Embarcadero application development products nicely supplement the Embarcadero database tools. Here, C++Builder supports three principle standards for access to the data sources:

- Borland Database Engine (BDE)
- ADO
- dbExpress

Each of these access methods is supported with VCL components for managing the connection with the respective data source, e.g. a SQL server, and the access to tables, queries and procedures.

DATA ACCESS WITH THE BDE

For compatibility reasons, the current versions still also support the BDE for access to local data in the dBASE or Paradox format. The respective VCL components are located in the "BDE" area of the Tool Palette.

Component	Description
TTable	VCL component for direct access to a table in the database.
TQuery	VCL component for data access via a SQL statement, with parameters and the result set which is based on the statement.
TStoredProc	VCL component for data access with a stored procedure, with parameters and the result set which is returned by the procedure.
TDatabase	VCL component for connection to the database, including application-specific parameters (via a BDE alias) and transaction control.
TSession	VCL component for the global management of database connections in an application.
TBatchMove	VCL auxiliary component for executing database operations (inserting, deleting, copying) on data record groups or complete tables.
TUpdateSQL	VCL auxiliary component for committing cached updates of "read only" queries or stored procedures to the database.
TNestedTable	VCL component for access to table data which exist as a field in another table. This component is possible only for access to special databases (Oracle as of version 8). A suitable BDE driver must be available.

Table 15: VCL components for data access via the BDE

As there is no further development here, you should no longer use these VCL components. If you have worked with data in xBase format up until now, there are some interesting alternatives

e.g. the Sybase Advantage Database Server. VCL components are also available for this and they can be fully integrated in the development environment.

DATA ACCESS WITH ADO

ADO (ActiveX Data Objects) is a Microsoft COM library for access to any data sources that support OLE DB. Alongside a multitude of databases, this includes objects similar to tables, e.g. the active directory of the operating system or the file system. This way, you can integrate a multitude of independent data sources in the programs that have been created with C++Builder. All database manufacturers which are to be taken seriously provide respective OLE DB drivers for their systems.

The VCL components for access via ADO are shown in the category "dbGo".

Component	Description
TADOConnection	This VCL component manages access to any ADO data source and provides parameters and methods (e.g. transactions, meta data).
TADOCommand	This VCL component is used for data access via an ADO command object. This component has parameters, but no result set. Used for SQL DML commands like "Insert", "Delete" and "Update".
TADODataSet	This VCL component is used to access a retrieved data set. This is a generic component which accesses tables directly or the data via an SQL query.
TADOTable	VCL component for direct access to tables. The component has a result set.
TADOQuery	This VCL component is used to access the data via a SQL query. The component has parameters and a result set. (For compatibility reasons, for BDE access, DML or DDL commands can be executed via a function. However, it is better to use "TADOCommand" here.)
TADOStoredProc	This VCL component is used to access the data via a stored procedure. This component can return either individual values or result sets. The component has parameters and a result set.
TRDSConnection	This VCL component is used to map a RDS DataSpace object in ADO. DataSpace objects are client-side proxies for access to business objects in a multitier application and are responsible for marshalling.

Table 16: VCL components for data access via the ADO

The Windows operating system is already equipped with the necessary support for the use of ADO. Access is controlled by means of a simple character string and this is known as the "ConnectionString".

DATA ACCESS WITH DBEXPRESS

The third variant, dbExpress, concerns a proprietary part of the VCL framework that has been written in Delphi and using which you can directly access the client interfaces of the database servers Oracle, Microsoft® SQL Server, DB2, Sybase, InterBase, MySQL and Firebird. This does have some speed advantages, but restricts the possible data sources. Moreover, data access is generally only unidirectional and the necessary interim drivers must be provided with the application.

To set up a data connection via dbExpress you must have an appropriate driver and two configuration files. The first file (dbxdrivers.ini) contains all the installed driver types (InterBase, Oracle, DB2, MSSQL, etc.) as well as the respective dynamic libraries required for each driver. This file also has the defaults for all the connection parameters. The second file (dbxconnections.ini) contains named connection configurations.

The VCL components for access via dbExpress are shown in the category "dbExpress" of the Tool Palette.

Component	Description
TSQLConnection	This VCL component is used to manage the access to database servers with a dbExpress driver. The component provides parameters and methods (e.g. transactions, meta data).
TSQLDataSet	This VCL component is used for unidirectional access to the data from a dbExpress data source. This way, you can call data from a table, a query or a procedure. Contrary to the other variants, here you can also carry out commands which do not return a data set (e.g. CREATE or INSERT). As access is unidirectional, no data is buffered. This restricts the navigation and filtering is not possible.
TSQLQuery	This VCL component is used for the unidirectional access to a query from a dbExpress data source. Statements which return no data set are also possible. As access is unidirectional, no data is buffered. This restricts the navigation and filtering is not possible.
TSQLStoredProc	This VCL component is used to access a saved procedure in a dbExpress data source. As access to the data set is unidirectional, no data is buffered. This restricts the navigation and filtering is not possible.
TSQLTable	This VCL component is used to access a stored procedure in a dbExpress data source. As access to the data set is unidirectional, no data is buffered. This restricts the navigation and filtering is not possible.
TSqlServerMethod	This VCL component is used to call DataSnap server methods.
TSQLMonitor	This VCL component is used to catch and save messages between a dbExpress data source and a data connection.

	This logs all SQL statements.
TSimpleDataSet	These are VCL components which retrieve data from a dbExpress data source and manage it in an internal cache. This adds unidirectional access and navigation possibilities to the speed advantage. Changes are returned to the data source.

Table 17: VCL components for data access via dbExpress

DATA-SENSITIVE VCL COMPONENTS AND DATA ACCESS

Alongside the VCL components for data access there is also a series of data-sensitive VCL components with which you can display and edit data promptly and without any great programming effort. In addition to the various input fields there is also a table representation (grid) and a button element (navigator). These are directly linked to a data source and already displayed with data at design time.

The respective VCL components are located in the "Data Controls" category of the Tool Palette.

Component	Description
TDBGrid	This VCL component is used as a data-sensitive control that displays the data of a data set in a table (grid). The data is displayed in the table and can be edited, deleted or inserted.
TDBNavigator	This VCL component is used to scroll in a data set and execute important operations (insert, delete, lock, update).
TDBText	This VCL component is a data-sensitive control in which the data of a field of a data set can be displayed. However, you cannot edit the data.
TDBEdit	This VCL component is a data-sensitive control in which the data of a field of a data set can be displayed and edited.
TDBMemo	This VCL component is a multi-line, data-sensitive control in which the data of a field of a data set can be displayed and edited. This component can be used to edit a field that contains an extensive amount of text.
TDBImage	This VCL component is a data-sensitive control in which graphics that are stored in at data set can be displayed (BLOB fields). The graphics can be copied, cut out, deleted or inserted via the Windows shortcuts.
TDBListBox	This VCL component is a data-sensitive control in which the data of a field of a data set can be displayed and, by selecting an entry from a list, edited.
TDBComboBox	This VCL component is a data-sensitive control in which the data of a field of a data set can be displayed and, by selecting an entry in a combination field (list or input field), edited.
TDBCheckBox	This VCL component is a data-sensitive control in which the data of a field of a data set can be enabled or disabled via a

	checkbox. This component is particularly suitable for Boolean values.
TDBRadioGroup	This VCL component is a data-sensitive control in which the data of a field of a data set in an option group can be displayed and edited. In doing so, only one value is valid at any time.
TDBLookupListBox	This VCL component is a data-sensitive control in which the data of a field of a data set can be displayed and, by selecting an entry from a list, edited. This list is defined by a different data set.
TDBLookupComboBox	This VCL component is a data-sensitive control in which the data of a field of a data set can be displayed and, by selecting an entry in a combination field, edited. This list of the combination field is defined by a different data set.
TDBRichEdit	This VCL component is a multi-line, data-sensitive control in which the data of an RTF field of a data set can be displayed and edited. This component is used to edit a field that contains an extensive amount of formatted text.
TDBCtrlGrid	This VCL component is a data-sensitive control that displays the data of a data set in a freely-definable format. Each data record is displayed in an area that is set once at design time.

Table 18: VCL components for data-sensitive controls

VCL components which are used to connect data sources ("TDataSource") with the data-sensitive controls described above are located in a different area. Other components are used as support when working with data (e.g. "TClientDataSet" for the local caching of data sets in the memory).

These are located in the area "Data Access" in the Tool Palette.

Component	Description
TDataSource	This VCL component is used to create a connection between a data set component and a data-sensitive control.
TClientDataSet	This VCL component is used to map an independent data set in the memory. The data can be read and saved from a different data source or a file. With the aid of a data set provider, access can also take place by means of a multitier architecture (remote data module).
TDataSetProvider	This VCL component is used to read data from a data source and returns updates to this. The component summarises the data of the source data set and transfers this in one or more data packages to a ClientDataSet or an XML broker. The ClientDataSet converts the data in the data package into a local copy and then manages this in the memory. Once the access has concluded, the ClientDataSet summarises the changed data and returns it to the provider. The provider commits the changes in the database or the ClientDataSet.
TXMLTransform	This VCL component is used to convert data between XML

	documents and data packages. This component can be used directly or together with TXMLTransformProvider or TXMLTransformClient. The auxiliary program "xmlmapper.exe" is used to create transformations.
TXMLTransformProvider	This VCL component is used to provide data from an XML document and commits the updates.
TXMLTransformClient	This VCL component is used to provide an adapter between an XML document and a provider.

Table 19: VCL components for data access (independent)

AN EXAMPLE DATABASE "TRAINING"

The following examples require a database. Unfortunately, many developers forget that programming with databases always starts with the creation of this database. It is particularly in this regard that many developers have considerable shortfalls and some declare (without any great theoretical basis) that normalization is no longer required these days. The data is usually saved "surface-orientated", i.e. in such a way that it will be displayed later. However, for an extensible application that needs to be adapted to new requirements over a long period, it is very important to have a clean structure of data. Many do not recognize the necessity of using efficient tools for this.

SETTING UP A "TRAINING" DATABASE

This example uses the Microsoft® SQL Server 2008 as a database backend. You can download an express version of this database free-of-charge from the Microsoft web site. The new database "Training" was set up for the following examples. This database manages people and addresses with the necessary value ranges. This database will be expanded to include employees and departments.

So, in the same way as an efficient development system is very important for programmers, the respective aids are also required for creating and managing a database. Embarcadero is an independent database tool provider. The management tool "DBArtisan" has been used to generate the database in this example. This program supports the database administrator in the management of various database systems (Oracle, IBM DB2, Sybase, MS SQL, MySQL, …) and can therefore also (but not only) be implemented in heterogeneous environments. A respective test version is available on the Embarcadero web site.

The following figure shows the creation of the database using a wizard on Microsoft® SQL server. This wizard guides the administrator through the set-up in several steps in which various parameters are entered. At the end, "DBArtisan" creates a SQL script which is then saved or directly executed.

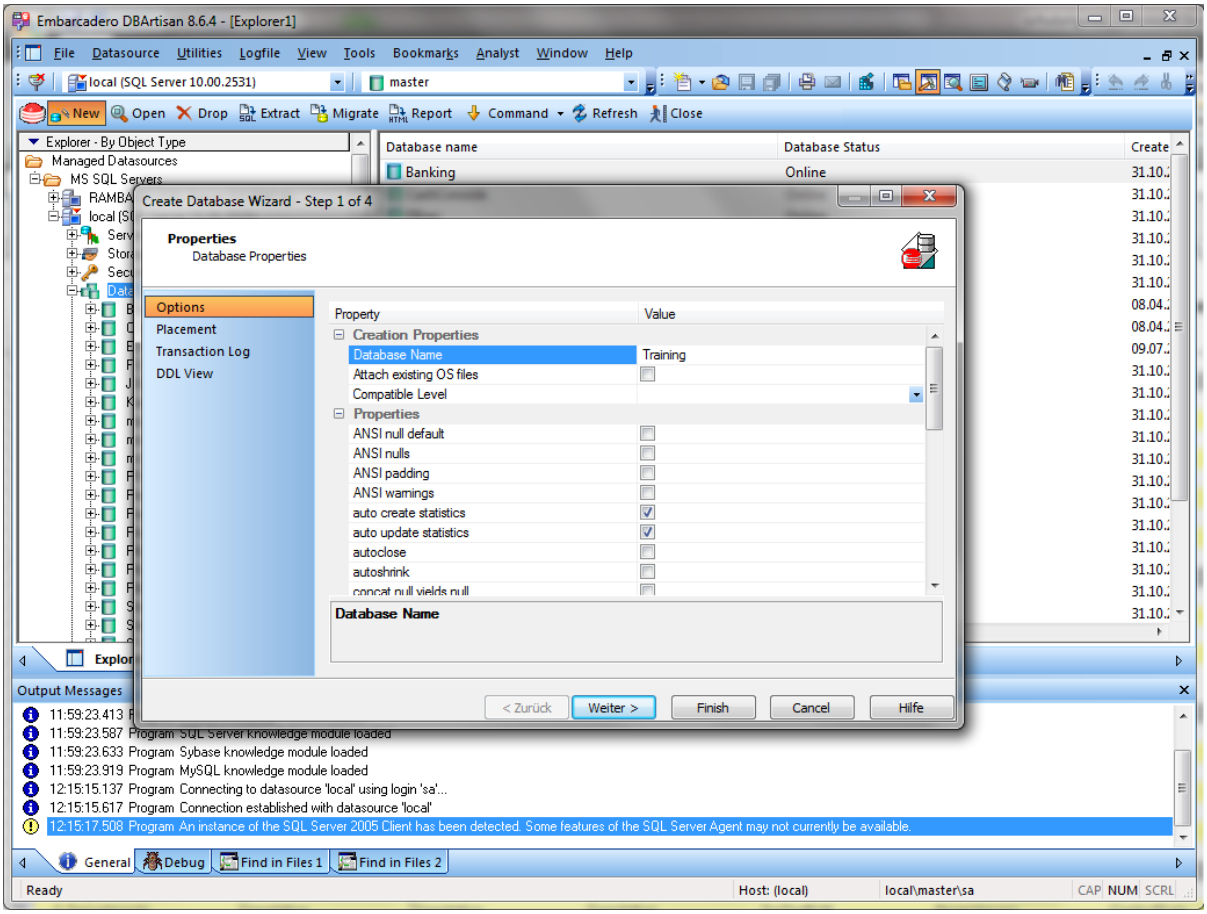

Figure 53: Setting up the "Training" database using DBArtisan

ESTABLISHING THE DATABASE STRUCTURE

The second step requires the database structure to be established. A modelling tool can be used for this (e.g. the ER/Studio) and the entries can be entered either directly in the respective interface or a SQL script containing the respective DDL commands is written. Modern design tools offer the option of creating a logical database model with the respective rules and various physical models for different SQL server backends. They also help with the documentation for the database and with discussions by means of the graphical representation. Embarcadero ER/Studio can be used for this purpose, and is also available as a free-of-charge test version to download from the Embarcadero web site.

The following listing contains the structure description of the "Training" database (other database servers could require deviating types, especially for free text and date values). The modelling tools mentioned above usually offer the option to read scripts or reorganize existing databases.

```
CREATE TABLE AddressTypes (
   ID           integer NOT NULL,
   Denotation   varchar(50) NOT NULL,
   Abbreviation varchar(10) NOT NULL,
   Description  text,
```

```
   UrgentValue  smallint
   );

CREATE TABLE Addresses (
   ID           integer NOT NULL,
   AddressType  integer NOT NULL,
   Street       varchar(50) NOT NULL,
   StreetNumber varchar(10),
   City         varchar(50),
   Zipcode      char(10),
   Country      integer
   );

CREATE TABLE Countries (
   ID           integer NOT NULL,
   Denotation   varchar(50) NOT NULL,
   Abbreviation varchar(10) NOT NULL,
   Description  text,
   UrgentValue  smallint,
   ISOCode      varchar(5)
   );

CREATE TABLE Departments (
   ID          integer NOT NULL,
   Denotation  varchar(30) NOT NULL,
   Description text
   );

CREATE TABLE Employees (
   ID                   integer NOT NULL,
   PersonnelNumber      varchar(15) NOT NULL,
   Salery               decimal(10, 2) NOT NULL,
   StartOfJob           datetime NOT NULL,
   JobPosition          integer,
   JobSpecification     text,
   Department           integer,
   SocialSecurityNumber varchar(20)
   );

CREATE TABLE FormsOfAddress (
   ID                integer NOT NULL,
   Denotation        varchar(30) NOT NULL,
   Abbreviation      varchar(10) NOT NULL,
   Description       text,
   UrgentValue       smallint,
   TypeSpecification integer NOT NULL,
   Salutation        varchar(50) NOT NULL,
   Valediction       varchar(50) NOT NULL
   );

CREATE TABLE JobPositions (
   ID           integer NOT NULL,
   Denotation   varchar(50) NOT NULL,
   Abbreviation varchar(10) NOT NULL,
   Description  text,
   UrgentValue  smallint
   );

CREATE TABLE PersonTypeSpecs (
   ID           integer NOT NULL,
   Denotation   varchar(50) NOT NULL,
   Abbreviation varchar(10) NOT NULL,
```

```
   Description  text,
   UrgentValue  smallint
   );

CREATE TABLE Persons (
   ID            integer NOT NULL,
   Name          varchar(30) NOT NULL,
   Firstname     varchar(30),
   FormOfAddress integer,
   Birthday      datetime,
   Notice        text
   );

ALTER TABLE AddressTypes ADD CONSTRAINT pk_AddressTypes
 PRIMARY KEY (ID);

ALTER TABLE Addresses ADD CONSTRAINT pk_Addresses
 PRIMARY KEY (ID, AddressType);

ALTER TABLE Countries ADD CONSTRAINT pk_Countries
 PRIMARY KEY (ID);

ALTER TABLE Departments ADD CONSTRAINT pk_Departments
 PRIMARY KEY (ID);

ALTER TABLE Employees ADD CONSTRAINT pk_Employees
 PRIMARY KEY (ID);

ALTER TABLE FormsOfAddress ADD CONSTRAINT pk_FormsOfAddress
 PRIMARY KEY (ID);

ALTER TABLE JobPositions ADD CONSTRAINT pk_JobPositions
 PRIMARY KEY (ID);

ALTER TABLE PersonTypeSpecs ADD CONSTRAINT pk_PersonTypeSpecs
 PRIMARY KEY (ID);

ALTER TABLE Persons ADD CONSTRAINT pk_Persons
 PRIMARY KEY (ID);

ALTER TABLE AddressTypes ADD CONSTRAINT uk_AddressTypes_2
 UNIQUE (Denotation);

ALTER TABLE AddressTypes ADD CONSTRAINT uk_AddressTypes_3
 UNIQUE (Abbreviation);

ALTER TABLE Countries ADD CONSTRAINT uk_Countries_2
 UNIQUE (Denotation);

ALTER TABLE Countries ADD CONSTRAINT uk_Countries_3
 UNIQUE (Abbreviation);

ALTER TABLE Countries ADD CONSTRAINT uk_Countries_4
 UNIQUE (ISOCode);

ALTER TABLE Departments ADD CONSTRAINT uk_Departments_2
 UNIQUE (Denotation);

ALTER TABLE Employees ADD CONSTRAINT uk_Employees_2
 UNIQUE (PersonnelNumber);

ALTER TABLE FormsOfAddress ADD CONSTRAINT uk_FormsOfAddress_2
```

```
 UNIQUE (Denotation);

ALTER TABLE FormsOfAddress ADD CONSTRAINT uk_FormsOfAddress_3
 UNIQUE (Abbreviation);

ALTER TABLE JobPositions ADD CONSTRAINT uk_JobPositions_2
 UNIQUE (Denotation);

ALTER TABLE JobPositions ADD CONSTRAINT uk_JobPositions_3
 UNIQUE (Abbreviation);

ALTER TABLE PersonTypeSpecs ADD CONSTRAINT uk_PersonTypeSpecs_2
 UNIQUE (Denotation);

ALTER TABLE PersonTypeSpecs ADD CONSTRAINT uk_PersonTypeSpecs_3
 UNIQUE (Abbreviation);

ALTER TABLE Addresses ADD CONSTRAINT fk_Addresses_Persons_Ref_1
 FOREIGN KEY (ID) REFERENCES Persons (ID);

ALTER TABLE Addresses ADD CONSTRAINT fk_Addresses_AddressTypes_Ref_2
 FOREIGN KEY (AddressType) REFERENCES AddressTypes (ID);

ALTER TABLE Addresses ADD CONSTRAINT fk_Addresses_Countries_Ref_3
 FOREIGN KEY (Country) REFERENCES Countries (ID);

ALTER TABLE Employees ADD CONSTRAINT fk_Employees_Persons_Ref_1
 FOREIGN KEY (ID) REFERENCES Persons (ID);

ALTER TABLE Employees ADD CONSTRAINT fk_Employees_JobPositions_Ref_2
 FOREIGN KEY (JobPosition) REFERENCES JobPositions (ID);

ALTER TABLE Employees ADD CONSTRAINT fk_Employees_Departments_Ref_3
 FOREIGN KEY (Department) REFERENCES Departments (ID);

ALTER TABLE FormsOfAddress ADD CONSTRAINT fk_FormsOfAddress_PersonTypeSpecs_Ref_1
 FOREIGN KEY (TypeSpecification) REFERENCES PersonTypeSpecs (ID);

ALTER TABLE Persons ADD CONSTRAINT fk_Persons_FormsOfAddress_Ref_1
 FOREIGN KEY (FormOfAddress) REFERENCES FormsOfAddress (ID);

CREATE INDEX idxAddresses_1 ON Addresses(Zipcode);

CREATE INDEX idxAddresses_2 ON Addresses(City);

CREATE INDEX idxAddresses_3 ON Addresses(Street, City);

CREATE INDEX idxEmployees_1 ON Employees(Department);

CREATE INDEX idxEmployees_2 ON Employees(SocialSecurityNumber);

CREATE INDEX idxEmployees_3 ON Employees(JobPosition);

CREATE INDEX idxPersons_1 ON Persons(Name, Firstname);

CREATE INDEX idxPersons_2 ON Persons(FormOfAddress);

CREATE INDEX idxPersons_3 ON Persons(Birthday);
```

Listing 12: SQL script for generating the database

You must now execute this script on the respective database server. The respective databases usually provide more or less suitable programs. Embarcadero can also offer an efficient, independent program for this - "RapidSQL". Again, you can use different data sources and there is a SQL syntax representation, a code completion, and the data is shown in tables where you can also edit it. You can save, print out or email the result sets.

The following figure shows "Rapid SQL" with the above script for generating the database.

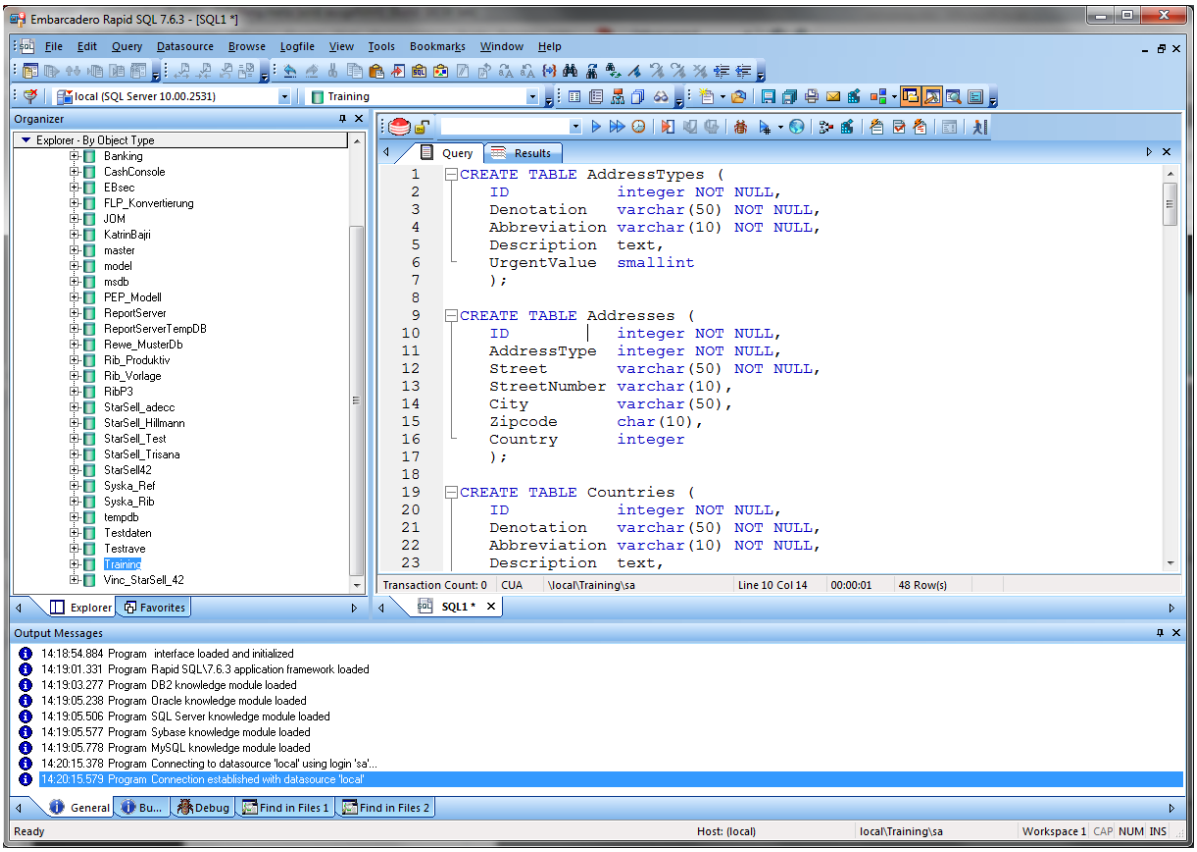

Figure 54: Executing the SQL script with "Rapid SQL"

ESTABLISHING THE VALUE RANGES

Having created the basic structure of the database, you must then set the value ranges in the following step. Also here you have the option to enter the data directly or to write a script. It is preferable to use the second option for serious projects, as this can always be repeated and, moreover, can also be a part of the documentation.

The following script establishes the value ranges for the application. The syntax may be slightly modified for a different database backend. The command "GO" is particularly typical of Microsoft.

```
/* ====================================================================== */
/* Fill the value tables with the necessary data                          */
/* ====================================================================== */
```

```
INSERT INTO PersonTypeSpecs (ID, Denotation, Abbreviation, UrgentValue)
VALUES (1, 'male', 'M', 1),
       (2, 'female', 'F', 1),
       (3, 'group',   'G', 1);
GO

INSERT INTO FormsOfAddress (ID, Denotation, Abbreviation, TypeSpecification,
                            Salutation, Valediction)
VALUES (1, 'Mr',     'Mr',  1, 'Dear Mr',            'Yours sincerely'),
       (2, 'Mrs',    'Mrs',  2, 'Dear Mrs',             'Yours sincerely'),
       (3, 'Family', 'Fam', 3, 'Dear family',          'Yours sincerely'),
       (4, 'Company', 'Co.', 3, 'Dear company',          'Yours sincerely'),

INSERT INTO AddressTypes (ID, Denotation, Abbreviation)
VALUES (1, 'Main address',     'AD'),
       (2, 'Billing address', 'RA'),
       (3, 'Delivery address', 'RA'),

GO

INSERT INTO Countries (ID, Denotation, Abbreviation, ISOCode)
VALUES (1, 'Federal Republic of Germany', 'BRD', 'DE');

INSERT INTO JobPositions (ID, Denotation, Abbreviation)
VALUES (1, 'Chairman',         'V'),
       (2, 'Head of department', 'AL'),
       (3, 'Head of group',    'GL'),
       (4, 'Employee',     'AN'),
       (5, 'Temporary worker',         'HK');
GO
```

Listing 13: Establishing the value ranges in the "Training" database

The following figure shows the "Training" database created so far as a logical model. Such models help the developer to detect associations more quickly and find necessary information. This saves time and avoids errors.

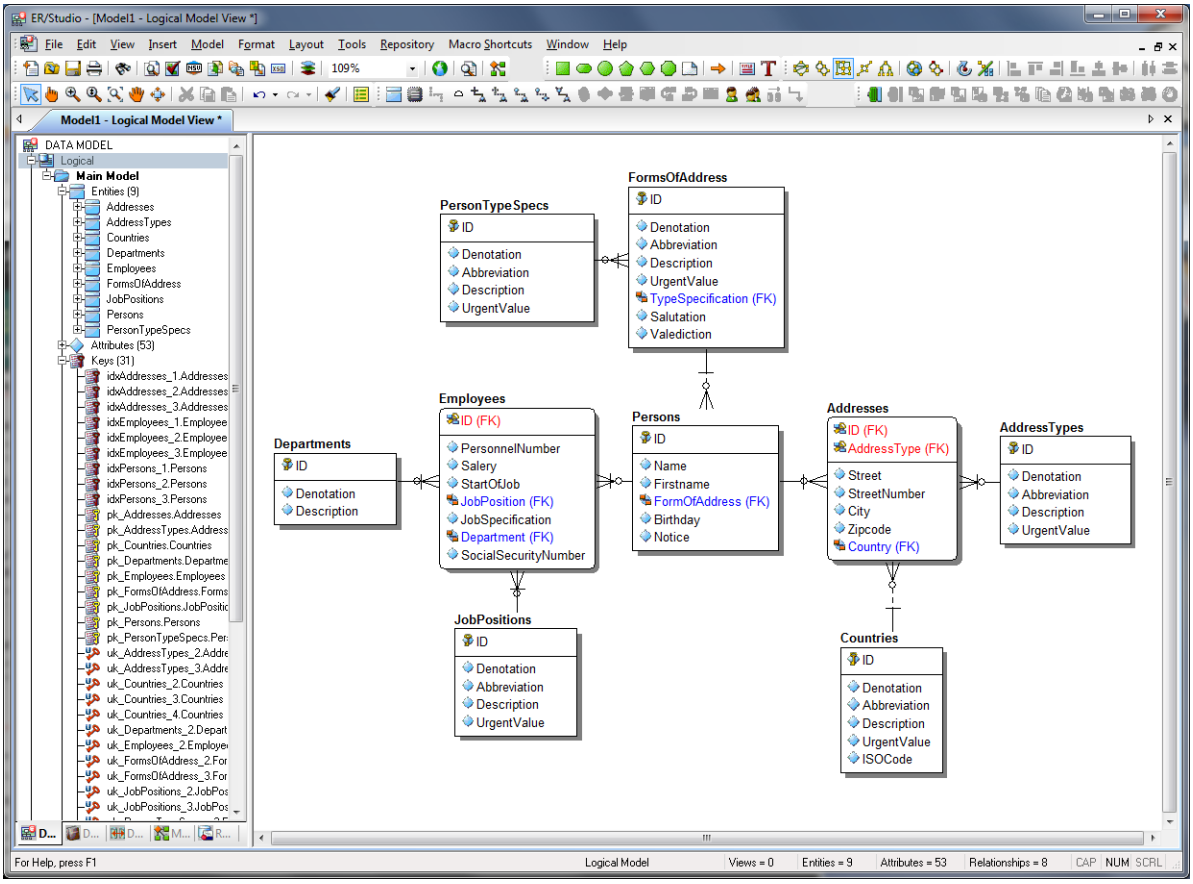

Figure 55: Portraying and editing the data model in "ER/Studio"

REFINING THE DATA MODEL

Many programmers would say now that it is much more difficult to access the normalised data. Some also maintain that the performance is worse. Such statements are usually the result of unfamiliarity with relational databases.

This model therefore defines two views for the access to the personal and employee data. As we would usually have to deal with more than one address in data records, the value table defines three different address types. In the first view “vwPersonenDaten", the respective records in the table "Addresses“ will be linked by a left join to the record from the table “Persons”. The majority of SQL servers are not just capable of reading clear views; they also allow to edit this data.

The second view summarises the data from the tables “Persons” and “Employees”, which have an is-a relationship (in programming this corresponds to a generalization), by means of a complete join. Then, the addresses are added as in the first view.

```
CREATE VIEW vwPersons (ID, Name, Firstnam, FormOfAddress, BirthDay, Notice,
                       AD_Street, AD_StreetNumber, AD_City, AD_Zipcode, AD_Country,
                       RA_Street, RA_StreetNumber, RA_City, RA_Zipcode, RA_Country,
                       LA_Street, LA_StreetNumber, LA_City, LA_Zipcode, LA_Country)
AS
   SELECT Ps.ID, Ps.Name, Ps.Firstname, Ps.FormOfAddress, ps.BirthDay, Ps.Notice,
```

```
          AD.Street AS AD_Street, AD.Street AS AD_StreetNumber, AD.City AS AD_City,
          AD.Zipcode AS AD_Zipcode, AD.Country AS AD_Country,
          RA.Street  AS RA_Street, RA.Street   AS RA_StreetNumber, RA.City AS RA_City,
          RA.Zipcode AS RA_Zipcode, RA.Country AS RA_Country,
          LA.Street  AS LA_Street, LA.Street   AS LA_StreetNumber, LA.City AS LA_City,
          LA.Zipcode AS LA_Zipcode, LA.Country AS LA_Country
   FROM   Persons Ps LEFT JOIN Addresses AD ON (AD.ID = Ps.ID AND AD.AddressType = 1)
                     LEFT JOIN Addresses RA ON (RA.ID = Ps.ID AND RA.AddressType = 2)
                     LEFT JOIN Addresses LA ON (LA.ID = Ps.ID AND LA.AddressType = 3);

CREATE VIEW vwEmployees (ID, Name, Firstnam, FormOfAddress, BirthDay, Notice,
                         PersonnelNumber, Salery, StartOfJob, JobPosition,
                         JobSpecification, Department, SocialSecurityNumber,
                         AD_Street, AD_StreetNumber, AD_City, AD_Zipcode, AD_Country,
                         RA_Street, RA_StreetNumber, RA_City, RA_Zipcode, RA_Country,
                         LA_Street, LA_StreetNumber, LA_City, LA_Zipcode, LA_Country)
AS
   SELECT Ps.ID, Ps.Name, Ps.Firstname, Ps.FormOfAddress, Ps.BirthDay, Ps.Notice,
          Es.PersonnelNumber, Es.Salery, Es.StartOfJob, Es.JobPosition,
          Es.JobSpecification, Es.Department, Es.SocialSecurityNumber,
          AD.Street  AS AD_Street, AD.Street   AS AD_StreetNumber, AD.City AS AD_City,
          AD.Zipcode AS AD_Zipcode, AD.Country AS AD_Country,
          RA.Street  AS RA_Street, RA.Street   AS RA_StreetNumber, RA.City AS RA_City,
          RA.Zipcode AS RA_Zipcode, RA.Country AS RA_Country,
          LA.Street  AS LA_Street, LA.Street   AS LA_StreetNumber, LA.City AS LA_City,
          LA.Zipcode AS LA_Zipcode, LA.Country AS LA_Country
   FROM   Persons Ps JOIN      Employees Es ON (Es.ID = Es.ID)
                     LEFT JOIN Addresses AD ON (AD.ID = Ps.ID AND AD.AddressType = 1)
                     LEFT JOIN Addresses RA ON (RA.ID = Ps.ID AND RA.AddressType = 2)
                     LEFT JOIN Addresses LA ON (LA.ID = Ps.ID AND LA.AddressType = 3);
```

Listing 14: Generating a view to access personal data

Finally, you specify in the database the respective privileges for the user. In principle, the author of the database possesses the necessary privileges. SQL servers provide a multitude of options for controlling privileges. The following examples assume that the necessary authorizations are available.

CREATING A PROGRAM FOR WORKING WITH THE DATA

The first step is to create a new VCL form application. This is explained in chapter "Creating a VCL form application" on page 50. The main form is saved under the name "MainForm.cpp", the project under the name "DatabaseApp".

To enter the following values for the VCL form, select the form in the Object Inspector.

Component	Description
Caption	Database test application
Name	frmMain

Table 20: Main form of the database application

The VCL framework has a special form type which is known as a data module. A data module has a design area but this is not visible. This is where you can "collect" invisible components, particularly the connection components for the data connections, but also the required dialogue components, image lists and the like.

To add a data module to the application, in the New Items dialogue you first select the area "C++Builder Files" from the tree view on the left side, and then the item "Data Module" from the entries on the right.

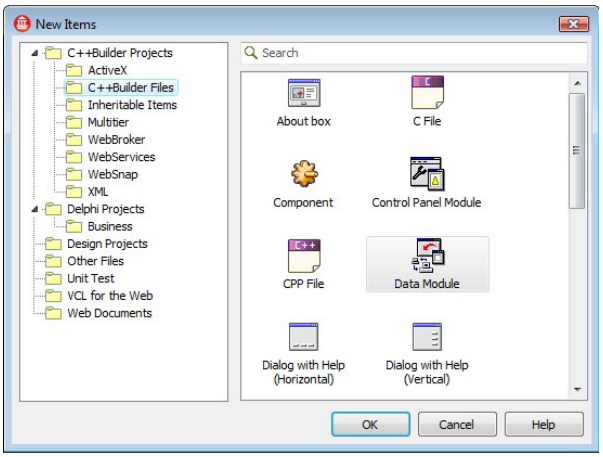

Figure 56: Creating a new data module via the New Items dialogue

In doing so, a data module is added to the project and in the IDE opened in design mode. In this example, the data module is saved as "DMUnit.cpp" and in the Object Inspector the name of it is changed to "DM".

GESTURE AND TOUCH CONTROL

Many will probably be familiar with the quote from Bjarne Stroustrup: "I have always wished for my computer to be as easy to use as my telephone; my wish has come true: because I can no longer figure out how to use my telephone." He said this in 1990, after attempting to use a multifunction telephone. We also know that it has become significantly easier to use multifunction telephones in recent years, and not least due to touch control. We are also familiar with interaction by touching the displays on various automats.

We have had simple tablet computers which only respond to a simple touch (usually also only by using a special stylus) for quite a while, and these were very expensive.

For many years now, Microsoft® has been working on a new, intuitive control for computers, the project "Surface" has surprised insiders for some time with its increasingly new, and, for many, at first very futuristic-sounding ideas. Microsoft® Windows Vista has already included a few touch extensions, e.g. a touch keyboard.

And now with Microsoft® Windows 7 we can see a multi-touch control in the broad area of the Windows computer. This has partially dissolved one of the oldest PC procedures - the interaction of the user by means of the keyboard and the mouse. It goes without saying that the hardware industry will respond very quickly with multi-touch-capable devices at favourable prices. With this in mind, we as developers must adapt our applications to meet these new technologies.

With C++Builder 2010 you are best equipped for this new challenge; as the only native C++ development environment it provides an integrated support for the touch and gesture control. In doing so, it supports the operating systems from Microsoft® Windows 2000 onwards, a large number of standard gestures are already specified and, alongside the touch (single, multi), it also supports styluses and the mouse.

Most of the VCL components have been appropriately enhanced so they directly support the gesture control. It also features a new VCL component with which you can add to your application a soft keyboard that supports the respective language settings.

IMPROVEMENTS TO THE NEW C++ STANDARD (C++0X) IN C++BUILDER 2010

The new C++ standard represents the first major improvement to the language since 1998. While 2003 saw an extension to the standard library, C++0x also made changes to the compiler and added a new library with the name "TR1".

One of the objectives of C++0x is to adapt to the new requirements for programming, e.g. distributed algorithms and the mutual use of memory between the various applications. A further objective was to make it easier to learn. And the most significant objective was for programs which had been correct in terms of the previous standard, to also be correct with respect to the new standard.

The new standard was originally planned for 2009, but this was delayed during the course of the year and we´re not really expecting anything before 2010. Having said that, the major compiler suppliers (including Embarcadero) have already implemented quite a lot of the standard for which consent has been gained from the Committee.

COMPILER ENHANCEMENTS

The following enhancements have already been implemented in C++Builder 2009 and 2010:

- New data type for the 64-bit integer: long long
  - long long value = 8446363626454LL;
- New literals
  - wstring strVal = u"København";
- RValues, move
  - int && ref;
  - more efficient handling of one-off data
  - argument is no longer used by the caller
- Type traits, static_assert
  - better error messages at compile time
- Type inference - decltype
  - corresponds to the result type of a statement
- Scoped enums
  - implicit cast to int prevented, perimeter of the namespace, possibility for forward declaration
- [[final]], [[noreturn]]

SCOPED ENUMS

With the enumerations there was previously the problem that the values of the enumerations had to have unique identifiers. It was therefore often the case that prefixes were written in front

(e.g. in the VCL) or the enumerations were hidden within classes (e.g. in the STL the ios class), so they only needed to have unique identifiers within the class.

```
enum WeekDay {   // alternatively with type e.g. enum Weekday: char { …
   Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };

std::string ToString(Weekday d) {
   switch(d) {
      case Weekday::Monday: return "Monday";    // possible use
      case Tuesday:           return "Tuesday";  // but not mandatory
       …
      }
   }

enum class Month {  // also with type specification, e.g. enum class Month: int { …
   January, February, March, April, May, June,
   July, August, September, October, November, December };

std::string ToString(Month m) {
   switch(m) {
      case Month::January: return "January";    // mandatory use
      case February:        return "February";   // error
       …
      }
   }
```

Listing 15: C++0x scoped enums

RVALUES

TYPE INFERENCE - DECLTYPE

The programming language C++ uses a strict type system. You are therefore always forced to clearly indicate the type of a variable. Unfortunately this isn´t always possible. This is why the new C++ standard gives you the option of exploiting the types of return values and using these for the establishment of a specific type. This way, you can generate a certain "type independence" in many methods.

```
#include <iostream>
#include <typeinfo>

using namespace std;

template <typename T1, typename T2>
void Addiere(T1 t1, T2 t2) {
   typedef decltype(t1 + t2) sumtype;
   sumtype t = t1 + t2;
   cout << "[" << typeid(t1).name() << "] t1 = " << t1 << endl
        << "[" << typeid(t2).name() << "] t2 = " << t2 << endl
```

```
        << "[" << typeid(t ).name() << "] t  = " << t  << endl
        << endl;
   return;
   }

int main(void) {
   Addiere( 5, 7);
   Addiere( 8, 3.14);
   Addiere("Hello ", "World");
   return 0;
   }
```

Listing 166: C++0x – type inference with decltype

IMPROVED CONTROL

An objective of the new standard is to improve the quality of the programs and increase the stability against errors. The command "static_assert" together with the methods known as "type_straits" play a key role here.

```
#include <iostream>
#include <type_traits>
class TTest;
class TTest2; // derived from TTest

template <typename T> void Polymorphic(T* t) {
   static_assert(std::tr1::is_polymorphic<T>::value, "must be a polymorphic type");
   };
template <typename T> void HasVirtualDestructor(T* t) {
   static_assert(std::tr1::has_virtual_destructor<T>::value,
                 "must have a virtual destructor");
   };

int main(void) {
   TTest testInstance;
   HasVirtualDestructor(&testInstance);
   Polymorphic(&testInstance);
   std::cout << "is_base_of<base, base> == " << std::boolalpha
             << std::tr1::is_base_of<TTest, TTest>::value << std::endl;
   std::cout << "is_base_of<base, derived> == " << std::boolalpha
             << std::tr1::is_base_of<TTest, TTest2>::value << std::endl;
   std::cout << "is_base_of<derived, base> == " << std::boolalpha
             << std::tr1::is_base_of<TTest2, TTest>::value << std::endl;
   return 0;
   };
```

Listing 17: C++0x – static-assert

THE BOOST LIBRARY IN C++BUILDER 2010

The Boost library is a collection of more than 70 sub-libraries which is available to you without restriction as a free library. Many of the libraries have been written by members of the Standards Committee, and this was stimulated by Robert Klarer and Beman Dawes. The idea for this library had its origin in 1998 when the first ISO standard for C++ was agreed. The name of the library refers to the saying "Boost is better than java" (Schnaps is better than coffee). The objective was to increase the productivity of C++ programmers and close gaps in the standard. And this on key platforms on which C++ compilers are available.

As was already the case with the standard library "STL", a lot of the work in Boost has been carried out using meta programming. Within the scope of the standardisation of C++, some of these libraries have at present been declared as meeting the official C++ standard in the Technical Report 1 (TR1); and Technical Report 2 (TR2) intends for more to follow.

The following table contains some of the important libraries and a brief explanation:

Library	Description
Any	Safe, generic container for individual values of different types.
Tupel	Return of several function values.
Threads	Portable multithreaded.
Interprocess	Shared memory, memory mapped files, process-shared mutexes, …
Date Time	Library for date and time based on generic program concepts.
Filesystem	Portable functions for querying and changing the path, files and directories.
Lexical Cast	Conversion of values to character strings and vice-versa, implemented as cast operator.
Format	Supports the formatting of the arguments via a format string, similar printf, but with 2 differences: Use of streams, type safe and support of self-defined types.
asio	Network programming, including sockets, timers, name resolution and socket io streams.
Serialization	Serialisation for persistence and marshalling (portable text, xml, binary).
Tokenizer	Parsing of character strings in tokens.
Spirit	Parser framework for EBNF syntax.
Math	Various mathematical libraries.
Python	Framework for integrating python scripts in C++ programs.

Table 21: Important Boost sub-libraries

Version 1.39 is being partially supported at present (current version is 1.41). However, this library is also available for C++Builder 2007 (BCBport by Alisdair Meredith) and C++Builder 2009 (integrated version 1.35).

According to a statement from Embarcadero, the current version of C++Builder 2010 supports the following libraries without restriction:

- algorithm/minmax

- algorithm/string
- any
- array
- crc
- disjoint_sets
- format
- functional
- logic
- property_map
- signals
- static_assert
- system
- tokenizer
- tuple
- utility/swap

The following libraries are restricted, and this could cause problems:

- config
- conversion
- dynamic_bitset
- filesystem
- integer
- io
- optional
- timer
- type_traits
- regex
- functional/hash
- test
- math
- tr1
- mpl
- range
- function
- function_types
- unordered
- utility
- utility/enable_if
- iterator
- asio
- variant

- numeric/interval
- exception
- circular_buffer
- parameter
- date_time
- concept_check
- assign
- numeric/conversion
- typeof
- spirit
- gil
- thread

The other libraries are not supported and are not included in the installation.

EXAMPLE WITH LEXICAL CASTS

The following brief example using lexical_cast demonstrates the usability of Boost. Maybe you´ve often looked for a simple way to input different data. This is why this example is called SimpleInput. This example is given for good reason, as time and again developers state that they have to fall back on VCL routines in C++, as C++ has nothing comparable to offer.

Create the following form in C++Builder.

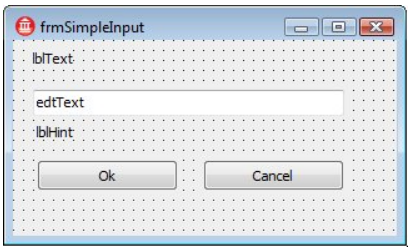

Figure 57: Form for the application "SimpleInput"

The structure of this dialogue is simple. Complete the input field, two labels and two buttons on the form and then name the figure accordingly.

Components	Property	Value
Components	Property	Value
frmSimpleInput	Caption	frmSimpleInput
	Name	frmSimpleInput
	Position	poMainFormCenter
lblText	Caption	lblText
	Name	lblText
edtText	Name	edtText
	Text	edtText
lblHint	Caption	lblHint

Components	Property	Value
Components	Property	Value
	Name	lblHint
btnOk	Caption	Ok
	Default	true
	ModalResult	mrOk
	Name	btnOk
btnCancel	Caption	Cancel
	ModalResult	mrCancel
	Name	btnCancel

Table 22: Properties of the VCL components in the "SimpleInput" program

The method "SimpleInput" actually controls the form. As we want to generate an input dialogue for different data types, we use this as a template.

The actual type cast is done in the operator lexical_cast, and, in the same way as the other cast operators, the target type is specified in the angle brackets.

```
//---------------------------------------------------------------------------
// Example program VCL dialogue and boost
// boost::lexical_cast and boost::format
// adecc Systemhaus GmbH
// 2009
//---------------------------------------------------------------------------
#ifndef SimpleInputH
#define SimpleInputH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>

#include <string>
#include <memory>

#include <boost/config.hpp>
#include <boost/config/compiler/CodeGear.hpp>
#include <boost/lexical_cast.hpp>
#include <boost/format.hpp>
//---------------------------------------------------------------------------
class TfrmSimpleInput : public TForm {
__published: // IDE-managed components
   TLabel  *lblText;
   TEdit   *edtText;
   TButton *btnOK;
   TButton *btnCancel;
   TLabel  *lblHint;
private:      // User declarations
public:       // User declarations
   __fastcall TfrmSimpleInput(TComponent* Owner);

};
```

```
//---------------------------------------------------------------------------
extern PACKAGE TfrmSimpleInput *frmSimpleInput;
//---------------------------------------------------------------------------
const std::string strMessageFormat = "%1%\n%2%\n%3% expected.";
template <typename T>
bool SimpleInput(std::string const& strCaption, std::string const& strText,
                 std::string const& strHint, T& theValue) {
   std::auto_ptr<TfrmSimpleInput> frm (new TfrmSimpleInput(0));
   frm->Caption             = strCaption.c_str();
   frm->lblText->Caption    = strText.c_str();
   frm->lblHint->Caption = strHint.c_str();
   frm->edtText->Text       = "";

   if(frm->ShowModal() == mrOk) {
      try {
         theValue = boost::lexical_cast<T>(frm->edtText->Text.c_str());
         }
      catch(std::exception& ex) {
         std::ostrstream os;
         os << boost::format(strMessageFormat) % ex.what()
                                               % frm->edtText->Text.c_str()
                                               % typeid(T).name()
            << std::ends;
         ShowMessage(os.str());
         }
      return true;
      }
   return false;
   }
```

Listing 18: The "TfrmSimpleInput" class and the "SimpleInput" method

If the type cannot be casted, the boost::lexical_cast triggers a standard exception. Boost::format is used within the handling. This routine is reminiscent of the old C output routines. The wildcards (%1%, %2%, …) are defined by means of a format string. Any type of wildcard and any order of sequence in the text are permitted. The use takes place within streams; the format and the parameters are separated by means of the character "%".

LIST OF TABLES

LIST OF FIGURES

LIST OF SOURCE TEXT

About the Author

Volker Hillmann is a founding director of the Berlin based software company adecc Systemhaus GmbH, where he is a passionate and very experienced C++ developer, software architect and database specialist. Volker is the author of "Object-oriented programming with C++" from Markt & Technik-Verlag, and is well known as a C++ and database trainer, as well as a speaker at conferences and industry events. Volker has a mathematics degree with specialized "databases and data security" and has been developing applications using C and C++ since 1988. adecc Systemhaus GmbH develops systems for insurance and financial services using C++Builder and also offers training courses for C/C++, C++Builder, and for working with databases.

EMBARCADERO TECHNOLOGIES.

Embarcadero Technologies, Inc. is a leading provider of award-winning tools for application developers and database professionals so they can design systems right, build them faster and run them better, regardless of their platform or programming language. Ninety of the Fortune 100 and an active community of more than three million users worldwide rely on Embarcadero products to increase productivity, reduce costs, simplify change management and compliance and accelerate innovation. The company's flagship tools include: Embarcadero® Change Manager™, Embarcadero RAD Studio, DBArtisan®, Delphi®, ER/Studio®, JBuilder® and Rapid SQL®. Founded in 1993, Embarcadero is headquartered in San Francisco, with offices located around the world. Embarcadero is online at www.embarcadero.com.