



Product Documentation

---

# Embarcadero®

## Delphi and C++Builder

### Quick Start Tutorial

XE Release  
PDF Last Published October 18, 2010

---

This PDF was generated from the docwiki content at:  
<http://docwiki.embarcadero.com/RADStudio/en/Tutorials>  
Some links in this PDF will refer to that website.  
Please send comments to: [documentation@embarcadero.com](mailto:documentation@embarcadero.com)  
PDF generated using the open source mwlib toolkit. See <http://code.pediapress.com/> for more information.

© 2010 Embarcadero Technologies, Inc. Embarcadero, the Embarcadero Technologies logos, and all other Embarcadero Technologies product or service names are trademarks or registered trademarks of Embarcadero Technologies, Inc. All other trademarks are property of their respective owners.

This software/documentation contains proprietary information of Embarcadero Technologies, Inc.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

If this software/documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

If this software/documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with Restricted Rights, as defined in FAR 552.227-14, Rights in Data-General, including Alternate III (June 1987).

Information in this document is subject to change without notice. Revisions may be issued to advise of such changes and additions. Embarcadero Technologies, Inc. does not warrant that this documentation is error-free.

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™, CodeGear™ 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](http://www.embarcadero.com).

ASIA-PACIFIC HEADQUARTERS  
L7, 313 LA TROBE STREET  
MELBOURNE VIC 3000  
AUSTRALIA

CORPORATE HEADQUARTERS  
100 CALIFORNIA STREET, 12TH FLOOR  
SAN FRANCISCO, CALIFORNIA 94111

EMEA HEADQUARTERS  
YORK HOUSE  
18 YORK ROAD  
MAIDENHEAD, BERKSHIRE  
SL6 1SF, UNITED KINGDOM

# Tutorial: Using the IDE for Delphi and C++Builder

---

This tutorial shows how to use the Delphi and C++Builder IDE to create applications.

## Topics

- **Introduction**
  - What is RAD Studio?
  - Finding information
- **Tour of the IDE**
  - First Look and Welcome Page
  - Toolbars
  - Tools
  - History Manager
  - Subversion Integration in the IDE
  - Data Explorer
  - Object Repository
  - IDE Insight
- **Starting your first RAD Studio application**
  - Using project templates from the Object Repository
  - Basic customization of the main form
  - Adding the components using the Form Designer
  - Customizing the components
  - Coding responses to user actions in the Code Editor
  - Compiling and running the application
  - Debugging the application
- **More advanced topics**
  - VCL and RTL
  - Third party add-ins
- **Other resources**
  - Embarcadero Developer Network
  - Partners

---

# Introduction Index (IDE Tutorial)

---

*Go Up to Tutorial: Using the IDE for Delphi and C++Builder*

This tutorial provides an overview of the Embarcadero RAD Studio development environment to get you started using the product right away. It also tells you where to look for details about the tools and features available in RAD Studio.

- Tour of the IDE describes the main tools on the Embarcadero RAD Studio desktop, or integrated development environment (IDE).
- Starting your first RAD Studio application explains how to use some of these tools to create an application.
- More advanced topics describes the more advanced features in RAD Studio, like VCL, RTL, or the included third party add-ins.
- Other resources displays a list of code-related articles on various products and the partners of Embarcadero.

For examples of specific functionality, see the Examples node of the Help or the Examples Wiki.

For sample projects that show how to write programs such as a text editor or database application, see the *Samples* directory of your Embarcadero RAD Studio installation. The samples are also accessible from **Start > Programs > Embarcadero RAD Studio > Samples**.

For a list of topics on using RAD Studio, see *Getting Started with RAD Studio*.

Other online resources are available at [www.embarcadero.com](http://www.embarcadero.com) <sup>[1]</sup>.

## Topics

- What is RAD Studio?
- Finding information

## References

[1] <http://www.embarcadero.com>

# What is RAD Studio?(IDE Tutorial)

---

*Go Up to Introduction Index (IDE Tutorial)*

Embarcadero RAD Studio is an object-oriented, visual programming environment for rapid application development (RAD). Using Embarcadero RAD Studio, you can create highly efficient visual applications with a minimum of manual coding, using either the Delphi, C++, or Delphi Prism programming languages. (To learn about using Prism to create .NET applications, see the Delphi Prism Primer <sup>[1]</sup>. For general information about Delphi Prism, see [prismwiki.embarcadero.com](http://prismwiki.embarcadero.com) <sup>[2]</sup>.)

RAD Studio provides all the tools you need to model applications, design user interfaces, automatically generate and edit code. It also gives you tools to compile, debug, and deploy applications. The tools available in the IDE depend on the version of RAD Studio you are using.

See What Is RAD Studio for more information.

## Next

Finding information

## References

[1] [http://prismwiki.embarcadero.com/en/The\\_Prism\\_Primer](http://prismwiki.embarcadero.com/en/The_Prism_Primer)

[2] <http://prismwiki.embarcadero.com>

# Finding information (IDE Tutorial)

---

*Go Up to Introduction Index (IDE Tutorial)*

You can find information about Embarcadero RAD Studio in the following ways:

- Web-based product documentation at [docwiki.embarcadero.com/RADStudio](http://docwiki.embarcadero.com/RADStudio) <sup>[1]</sup>
- F1 Help and Online Help System

**For information...**

about new features in this release, refer to [docwiki.embarcadero.com/RADStudio/en/What's\\_New](http://docwiki.embarcadero.com/RADStudio/en/What's_New) <sup>[2]</sup>.

## Web-based Product Information

You can get help online by visiting the [www.embarcadero.com](http://www.embarcadero.com) <sup>[1]</sup> web site and navigating to one of the following:

- Developer network—<http://dn.embarcadero.com> —where you can find news and articles about Embarcadero products.
- QualityCentral—<http://qc.embarcadero.com> —where you can read, create, update, or manage reports about issues in the Embarcadero products.
- CodeCentral—<http://cc.embarcadero.com> —where you can find, comment upon, upload, and download code snippets for the Embarcadero products.
- Blogs—<http://blogs.embarcadero.com> —where you can find useful information in articles written by the Embarcadero employees.

The [embarcadero.com](http://www.embarcadero.com) <sup>[1]</sup> web site also includes a list of books and additional technical documents for all of the Embarcadero products.

## F1 Help and Online Help System

You can get context-sensitive help in any part of the development environment, including in menu items, in dialog boxes, in toolbars, and in components by selecting the item and pressing F1.

Pressing the F1 key while a menu item is selected displays context-sensitive help for the item. For example, pressing F1 on the **Trace Into** menu item...

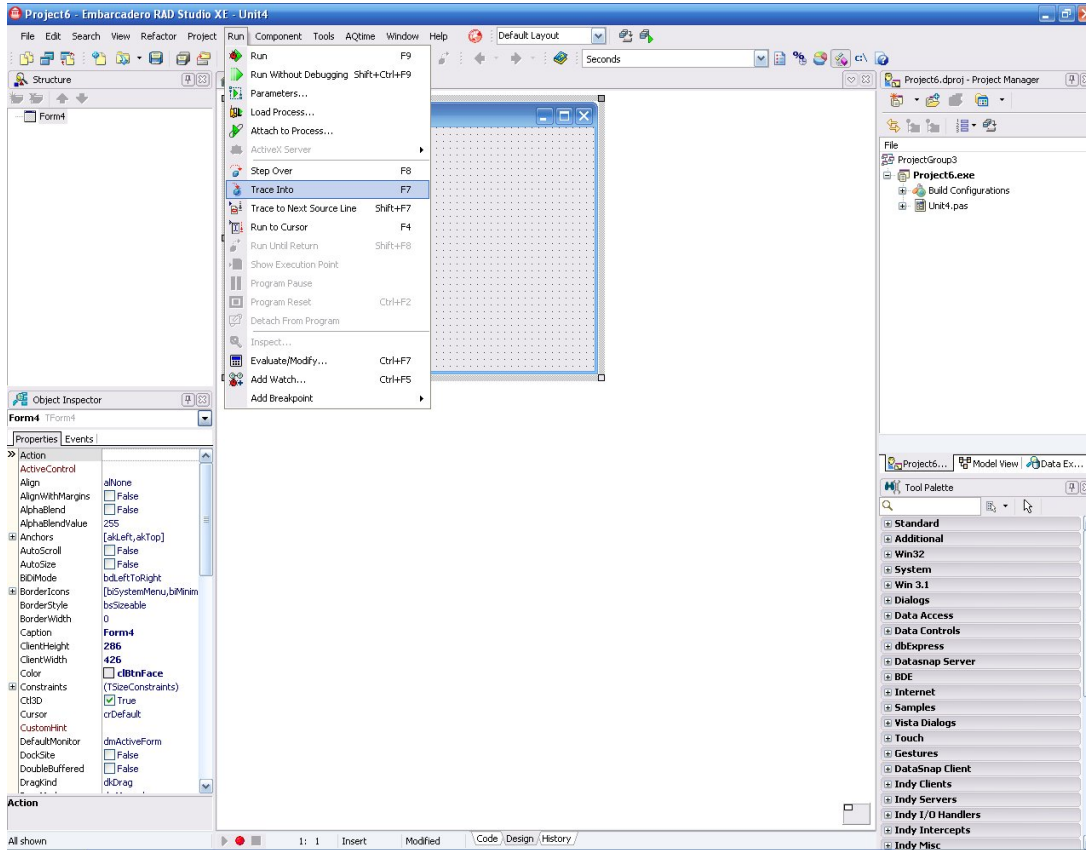


Figure 1-1. The **Trace Into** menu item under **Run** menu

...displays the following help page.

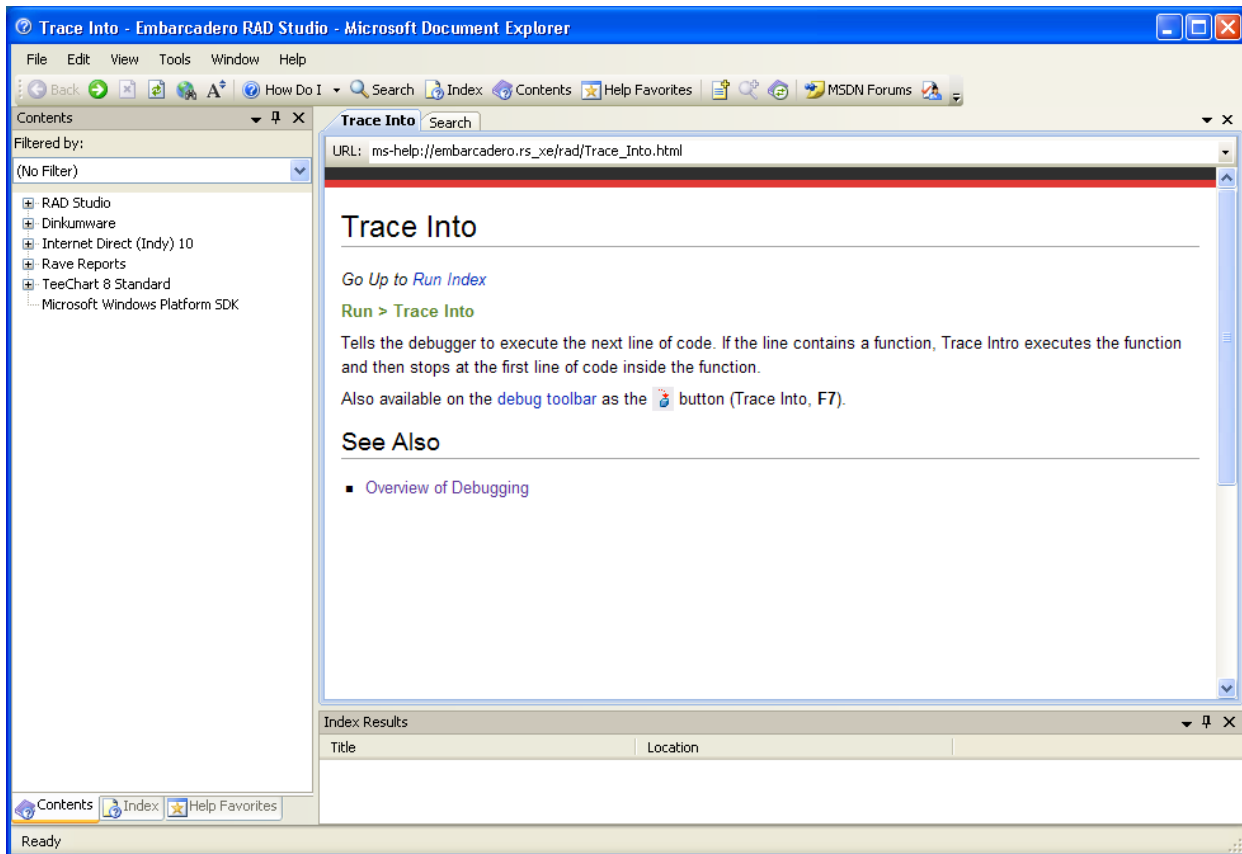


Figure 1-2. F1 Help for the **Trace Into** menu item

Error messages from the compiler and linker appear in a special window below the **Code Editor**. To get help with compilation errors, select a message from the list and press F1.

Useful information about using the help viewer can be found at <http://edn.embarcadero.com/article/37562>.

## Next

Tour of the IDE

- First Look and Welcome Page
- Toolbars
- Tools
- History Manager
- Data Explorer

## References

- [1] [http://docwiki.embarcadero.com/RADStudio/en/Main\\_Page](http://docwiki.embarcadero.com/RADStudio/en/Main_Page)
- [2] [http://docwiki.embarcadero.com/RADStudio/en/What%27s\\_New](http://docwiki.embarcadero.com/RADStudio/en/What%27s_New)

# Tour of the IDE Index (IDE Tutorial)

---

*Go Up to Tutorial: Using the IDE for Delphi and C++Builder*

**Note:** Also see the similarly named help topic **Tour of the IDE** for an overview of the elements of the IDE.

## Topics

- First Look and Welcome Page
- Toolbars
- Tools
  - Accessibility options
  - Form Designer
  - Tool Palette
  - Object Inspector
  - Project Manager
  - File Browser
  - Structure View
  - The Code Editor
    - Code Navigation
    - Formatting Source Code
    - Code Folding
    - Change Bars
    - Block Comments
    - Live Templates
    - SyncEdit
    - Code Insight
    - Refactoring
    - Keystroke Macros
    - To-Do Lists
- History Manager
- Subversion Integration in the IDE
- Data Explorer
- Object Repository
- IDE Insight



# First Look (IDE Tutorial)

*Go Up to Tour of the IDE Index (IDE Tutorial)*

When you start Embarcadero RAD Studio, the integrated development environment (IDE) launches and displays several tools and menus.

The IDE helps you visually design user interfaces, set object properties, write code, view, and manage your application in various ways.

The default IDE desktop layout includes some of the most commonly used tools. You can use the **View** menu to display or hide certain tools. You can also customize your desktop by moving or deleting elements. You can then save and use the desktop layouts that work best for you.

## Welcome Page

When you open RAD Studio, the *Welcome Page* appears with a number of links to developer resources, such as product-related articles, training, and online Help.

As you develop projects, you can quickly access them from the list of recent projects at the top of the page. To return to the **Welcome Page** from another main window such as the **Code Editor** or **Design** window, click the **Welcome Page** tab at the top of the window. If you close the Welcome Page, you can reopen it with the menu item **View > Welcome Page**.

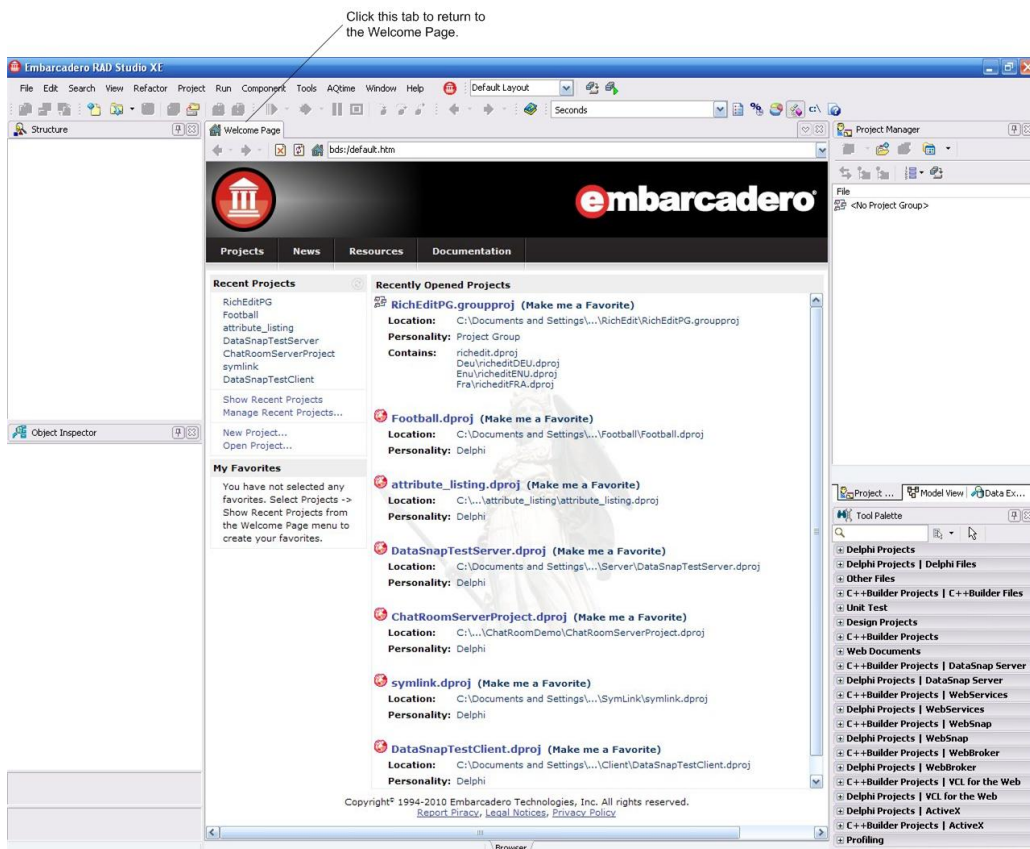


Figure 2-1. The RAD Studio Welcome Page

The following sections describe and show screenshots of the various available options when a RAD Studio project is open.

You can create a new project by simply clicking **File > New > VCL Forms Application - Delphi** for Delphi or **File > New > VCL Forms Application - C++Builder** for C++Builder. A more detailed explanation on how to create a

project is given in Starting your first RAD Studio application.

The following figure shows the IDE immediately after creating a new project that uses a VCL form:

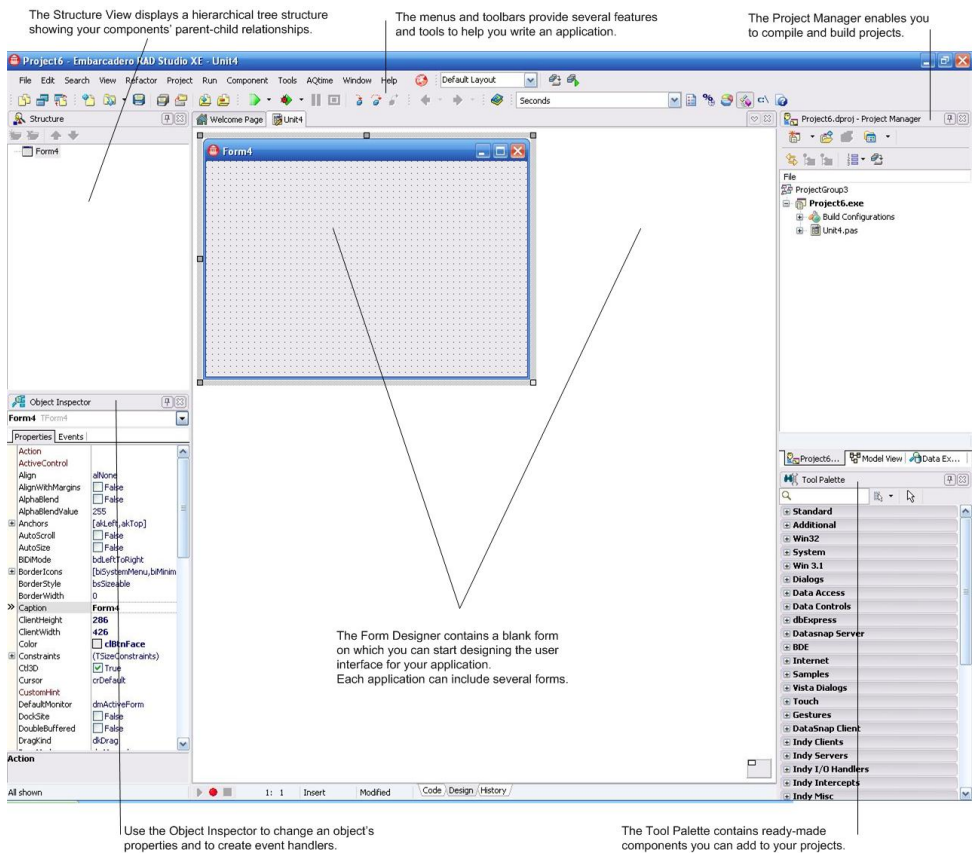


Figure 2-2. The default layout after creating a RAD Studio application

The main window, which occupies the top of the screen, contains the menu bar and the toolbars.

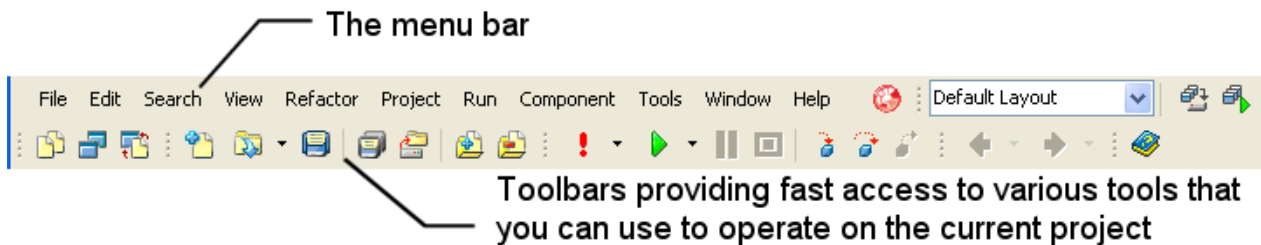


Figure 2-3. The main menu bar and toolbar

To control the toolbars that are displayed in the IDE, use **View > Toolbars**.

## Next

### Toolbars

# Toolbars (IDE Tutorial)

Go Up to Tour of the IDE Index (IDE Tutorial)

RAD Studio toolbars provide quick access to frequently used operations and commands. Some of the toolbars are displayed below in more detail. Most toolbar operations are duplicated in the drop-down menus.

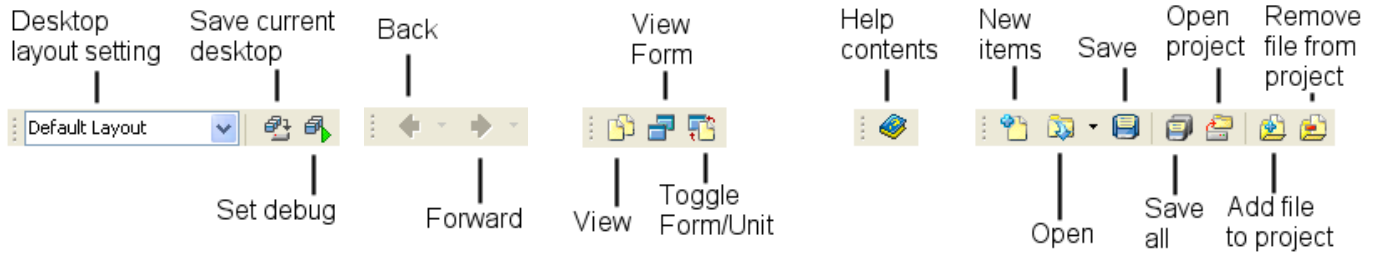


Figure 2-4. The main toolbars (Desktop, Browser, View, Help, Standard, and Debug toolbars)

To find out what a button does, hover the mouse over it for a moment until a tooltip appears. You can hide any toolbar by right-clicking the toolbar and selecting the context menu command **Hide**. To display a toolbar that is not showing, click **View > Toolbars** and check the toolbar you want.

Many operations have keyboard shortcuts as well as toolbar buttons. When a keyboard shortcut is available, the dropdown menu displays the shortcut next to the command.

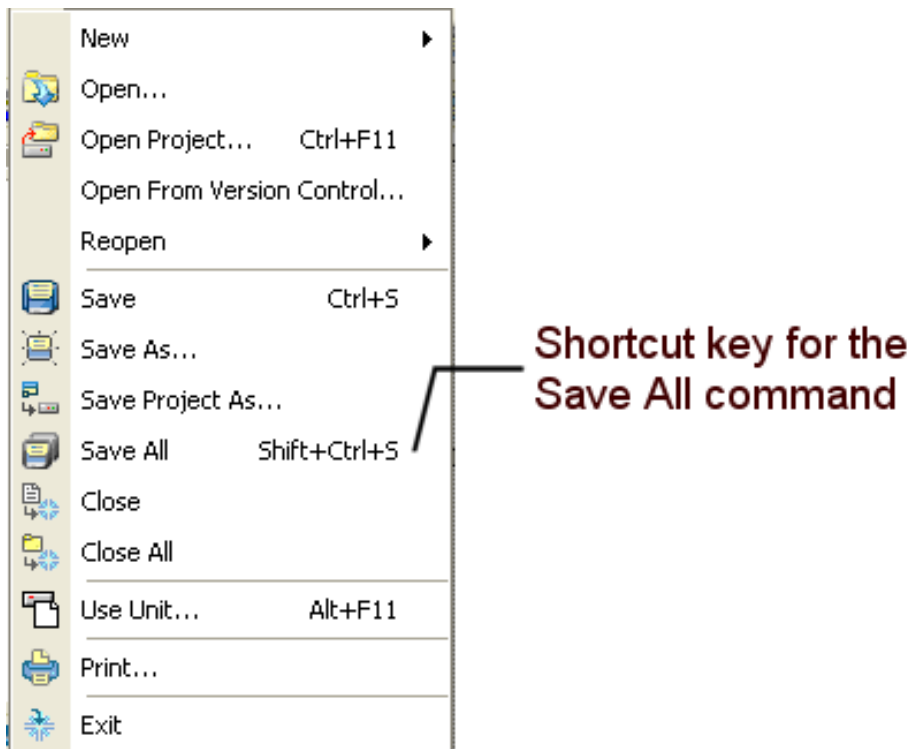


Figure 2-5. Shortcut keys in the File menu

You can right-click on many tools and icons to display a menu of commands appropriate to the object you are highlighting. These are called *context menus*. The toolbars are also customizable. You can add the commands you want to the toolbar or move the commands to different locations on the toolbar.

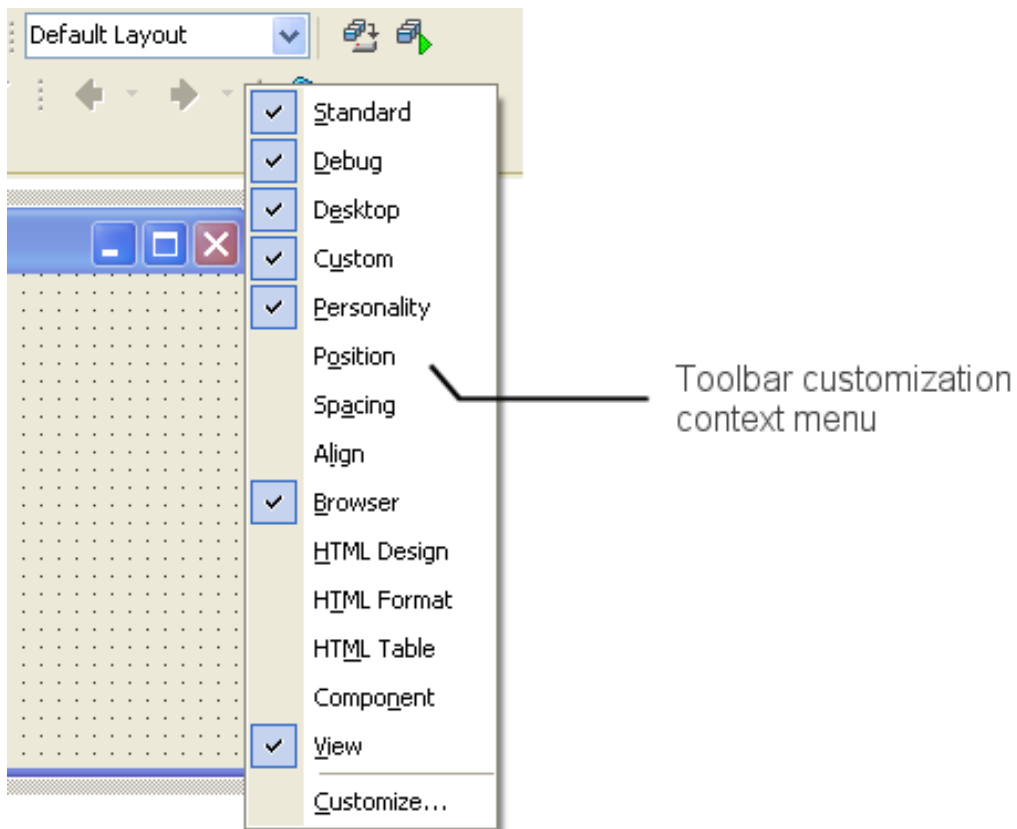


Figure 2-6. Customizing the toolbars

## Next

### Tools

- Accessibility options
- Form Designer
- Tool Palette
- Object Inspector
- Project Manager
- File Browser
- Structure View
- The Code Editor

# Tools Index (IDE Tutorial)

---

*Go Up to Tour of the IDE Index (IDE Tutorial)*

The tools available in the RAD Studio IDE depend on the version of RAD Studio you are using. However, every version of RAD Studio contains the tools described in the following topics.

## Topics

- Accessibility options
- Form Designer
- Tool Palette
- Object Inspector
- Project Manager
- File Browser
- Structure View
- The Code Editor
  - Code Navigation
  - Formatting Source Code
  - Code Folding
  - Change Bars
  - Block Comments
  - Live Templates
  - SyncEdit
  - Code Insight
  - Refactoring
  - Keystroke Macros
  - To-Do Lists

# Accessibility options (IDE Tutorial)

*Go Up to Tools Index (IDE Tutorial)*

The IDE's main menu supports MS Active Accessibility (MSAA). This means that you can use the Windows accessibility tools from the **Start** menu by choosing **All Programs > Accessories > Accessibility**.

These options are useful for people who have various disabilities, such as visual impairments. These options include such aids as being able to perform all functions with a keyboard only, having textual descriptions for images, not using color coding as the only means of conveying information, and modes of operation that do not require fine motor control.

## Next

Form Designer

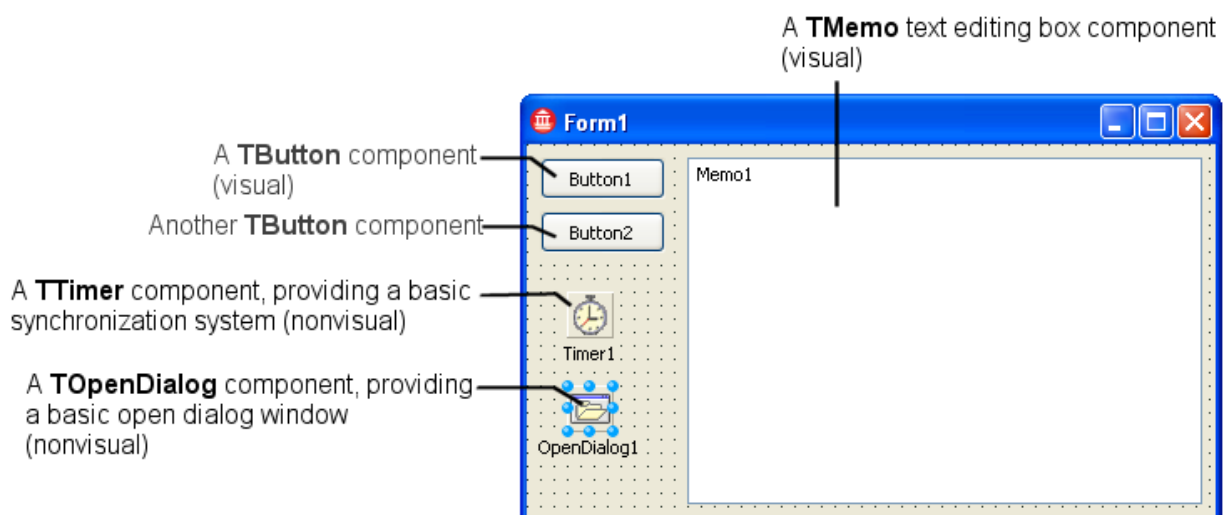
# Form Designer (IDE Tutorial)

*Go Up to Tools Index (IDE Tutorial)*

The *Form Designer* in RAD Studio allows you to rapidly prototype, build, and modify the user interface of your application. Typically, a form represents a window or an HTML page in the user interface.

Select the form that best suits your application design, whether it is a Web application that provides business logic functionality over the Web or a Windows application that provides processing and high-performance content display.

In RAD Studio, the user interface of an application is built using *components* that can be either visual or nonvisual and can be added to the form using the *Tool Palette*, discussed in the next section. Visual components appear on your form at the time the program is run. Nonvisual components, such as a file open dialog, do not appear on the form at run time, but provide capabilities to your application. Both types of components can be accessed at run time from your application's code.



*Figure 2-7. Creating a basic RAD Studio application using the Form Designer*

The RAD Studio **Form Designer** is based on the WYSIWYG (What You See Is What You Get) concept, allowing you to design your application's user interface with as little effort as possible.

In line with this concept, RAD Studio provides the capability of seeing how your application will behave at run time *before* you actually build and run the application. In other words, you can see how your application's components

operate at *design time* rather than run time, simplifying the debugging process.

Database-aware components allow you to do database queries and connections at design time. For instance, database viewing components can show data from a selected database. This way you can check at design time whether the behavior is the intended behavior or not.

The **Form Designer** also allows you to build user interfaces for VCL for Web applications, making development as simple as possible.

To start using the **Form Designer**, you must first create a VCL for the Web or a VCL for Win32 form using the project templates from the Object repository.

After you place components on the form, or **Form Designer**, you can arrange them the way they should look on your user interface. Every component's attributes can be viewed and changed with the *Object Inspector*. You can use the **Object Inspector** for many purposes, including the following:

- Set design-time properties for the components you place on the form.
- Create event handlers, filter-visible properties, and events, making the connection between your application's visual appearance and the code that makes your application run.

*For more information...*

See Adding the components using the Form Designer and Customizing the components in the next section.

## Next

Tool Palette

---

# Tool Palette (IDE Tutorial)

---

*Go Up to Tools Index (IDE Tutorial)*

The *Tool Palette* contains the components you use to develop your application. The **Tool Palette** is displayed as a list of component categories and is usually located in the IDE's right column. Each of the categories in the **Tool Palette** contains icons that represent visual or nonvisual components.

The categories divide the components into functional groups. For example, the Standard and Win32 categories include Windows controls such as a button, or an edit box. The Dialogs category includes common dialog boxes to use for file operations, such as opening and saving files. Click the plus sign on a category tab to see all the components in that category.

The contents of the **Tool Palette** change when switching between displaying the **Form designer** (*Design mode*) and displaying the application's code in the *Code Editor*. More information on the **Code Editor** is given in the Code Editor section. Thus, if you are viewing a form in Design mode, the **Tool Palette** displays components that are appropriate for a form.

You can also filter the components you see in the **Tool Palette**. When you type text in the filter field, only components whose name contains that string are listed. Pressing CTRL+Alt+P forces focus to the palette's filter.

Double-clicking a component on the **Tool Palette** adds it to your form. You can also drag a control to a desired position on the form. After components are on the form, you can drag them to the precise positions you want.

Each component has specific attributes—properties, events, and methods—that enable you to control your application. Use the *Object Inspector* to customize the components' behavior, as shown in the following section.

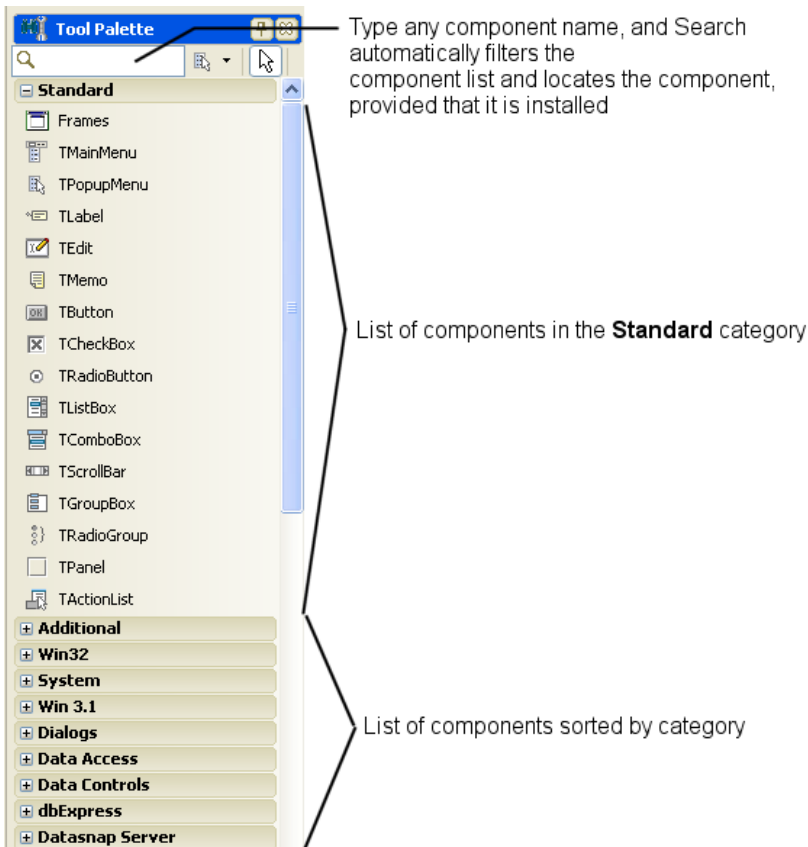


Figure 2-8. The Tool Palette showing the Standard components category

If you are viewing code in the **Code Editor**, the **Tool Palette** displays project types that you can add to your project group and file types that you can add to your project.

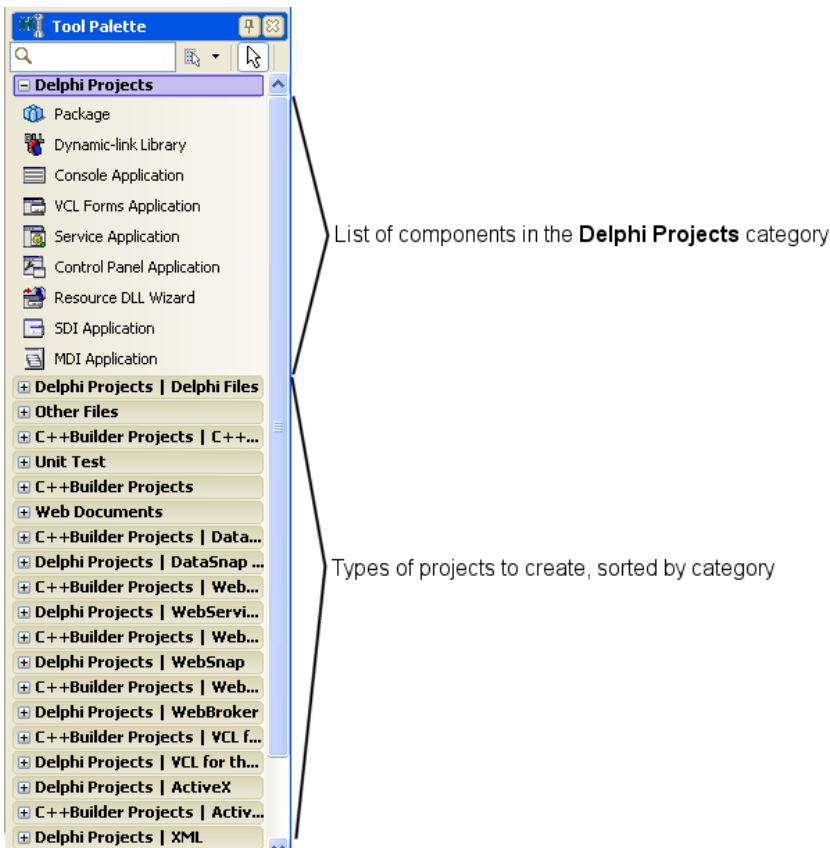


Figure 2-9. The Tool Palette in Code Editor mode, showing the standard Delphi Projects templates



You can customize the **Tool Palette**, changing the order of the categories by dragging them to a new location in the list. You can also create a new category and copy frequently used components into it.

In addition to the components that are installed with RAD Studio, you can add customized or third-party components to the **Tool Palette** and save them in their own category.

You can also create templates that are composed of one or more components. After arranging components on a form, setting their properties, and writing code for them, you can save them as a component template. Later, by selecting the template from the **Tool Palette**, you can place the preconfigured components on a form in a single step, which results in all associated properties and event-handling code being added to your project. You can reposition the components independently, reset their properties, and create or modify event handlers for them just as if you had placed each component in a separate operation.

## Next

Object Inspector

---

# Object Inspector (IDE Tutorial)

---

*Go Up to Tools Index (IDE Tutorial)*

The *Object Inspector* allows you to customize the properties of and create event handlers for the components in the application user interface. Each component has a set of published properties and events that the **Object Inspector** displays and allows to be modified visually, using the **Properties** and **Events** tabs.

User interfaces created with RAD Studio are *event-driven*, meaning that any component can react to an externally or internally generated event. Each component has a variety of events you can use as triggers to execute code. The **Object Inspector** allows you to automatically generate code that is executed when an event occurs. See [Creating an event handler](#) for details.

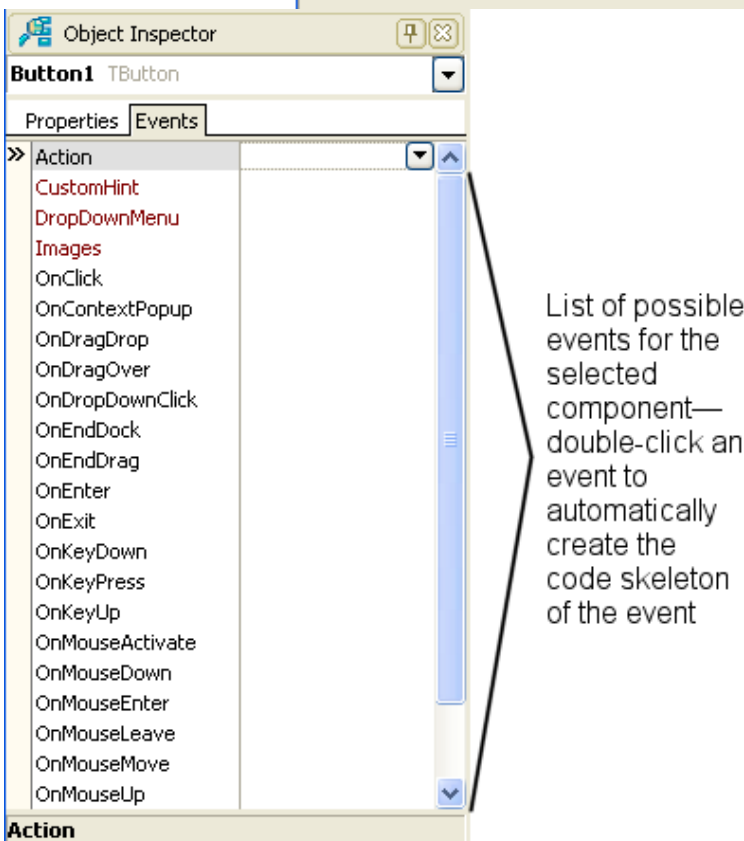
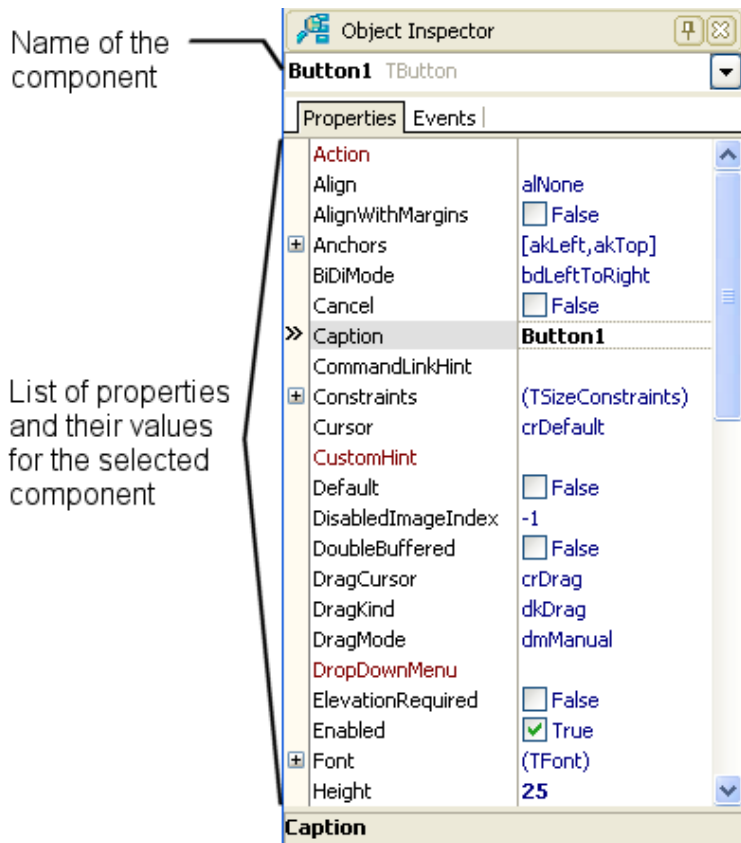


Figure 2-10. Properties tab in the Object Inspector

Figure 2-11. Events tab in the Object Inspector

You can customize the **Object Inspector** by right-clicking it. The context menu is shown with a list of customization options, such as the arrangement style of properties or the filtering options. See **Object Inspector** for descriptions of the context menu commands.

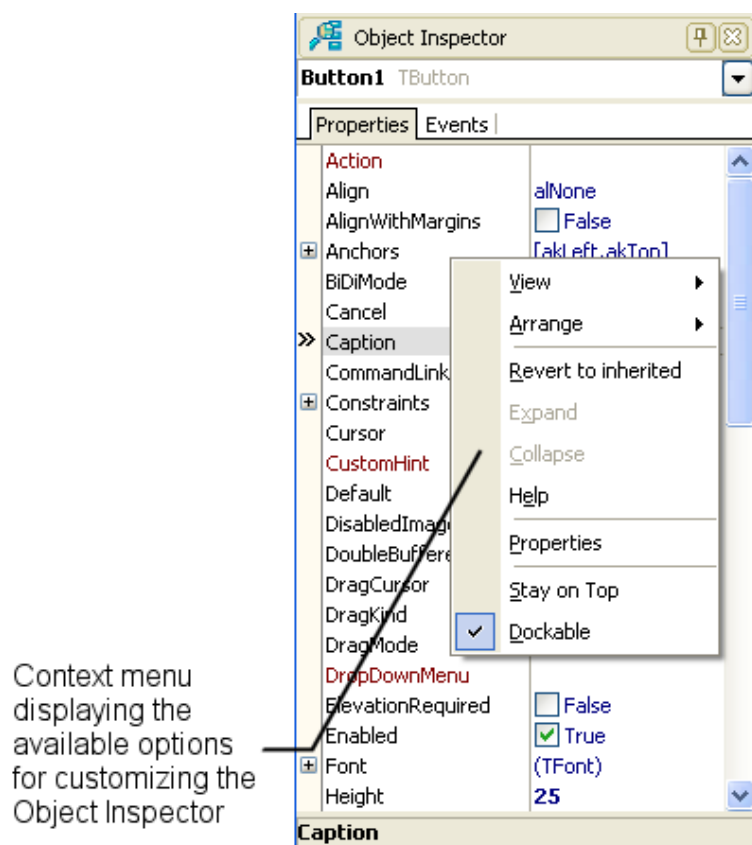


Figure 2-12. Customizing the Object Inspector

## Next

Project Manager

# Project Manager (IDE Tutorial)

*Go Up to Tools Index (IDE Tutorial)*

To build an application or a DLL using Delphi or C++Builder, you need to create a project. The *Project Manager* displays a hierarchical file list of your project or project group, so you can view and organize the files in the project or project group.

You can use the **Project Manager** to combine and display information on related projects into a single project group. By organizing related projects into a group, such as multiple executables, you can compile them at the same time. You can also set project options to resolve the dependencies between projects.

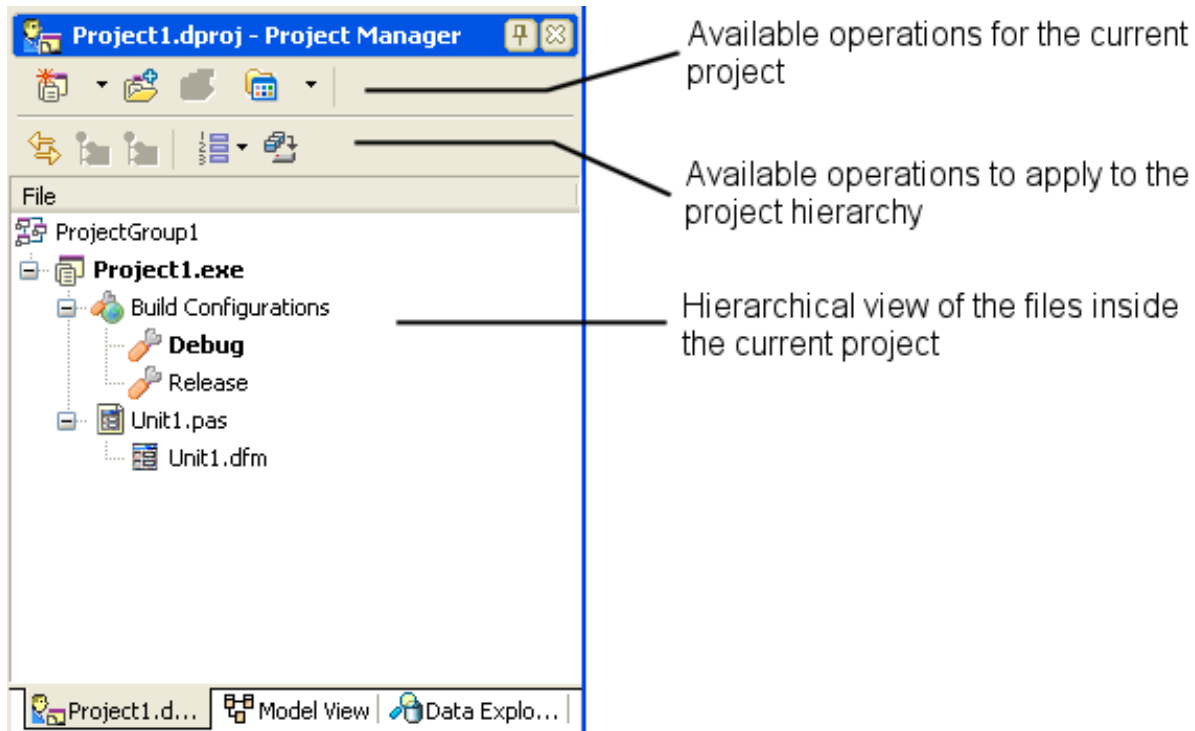


Figure 2-13. Hierarchical file list of the project, displayed by the Project Manager

The buttons at the top of the **Project Manager** enable you to perform the following tasks:

- **Activate** — Activate the currently selected project.
- **New** — Add another project to the current project group. If only one project currently exists, a project group is created for you automatically.
- **Remove** — Remove a project from the current project group.
- **View** — View the file tree hierarchy in multiple ways.
- **Sync** — Synchronize the project manager with the medium where the actual project or project group files are stored.
- **Expand** — Expand all child nodes of the currently selected node.
- **Collapse** — Collapse all child nodes of the currently selected node.
- **Sort** — Sort project (and set default sort order).
- **Build Groups** — Open Build Groups pane.

## Next

File Browser

# File Browser (IDE Tutorial)

*Go Up to Tools Index (IDE Tutorial)*

The *File Browser* allows you to conveniently manage files for a specified path. All files are displayed in a tree view for easy hierarchical browsing. To show the **File Browser**, choose **View > File Browser**.

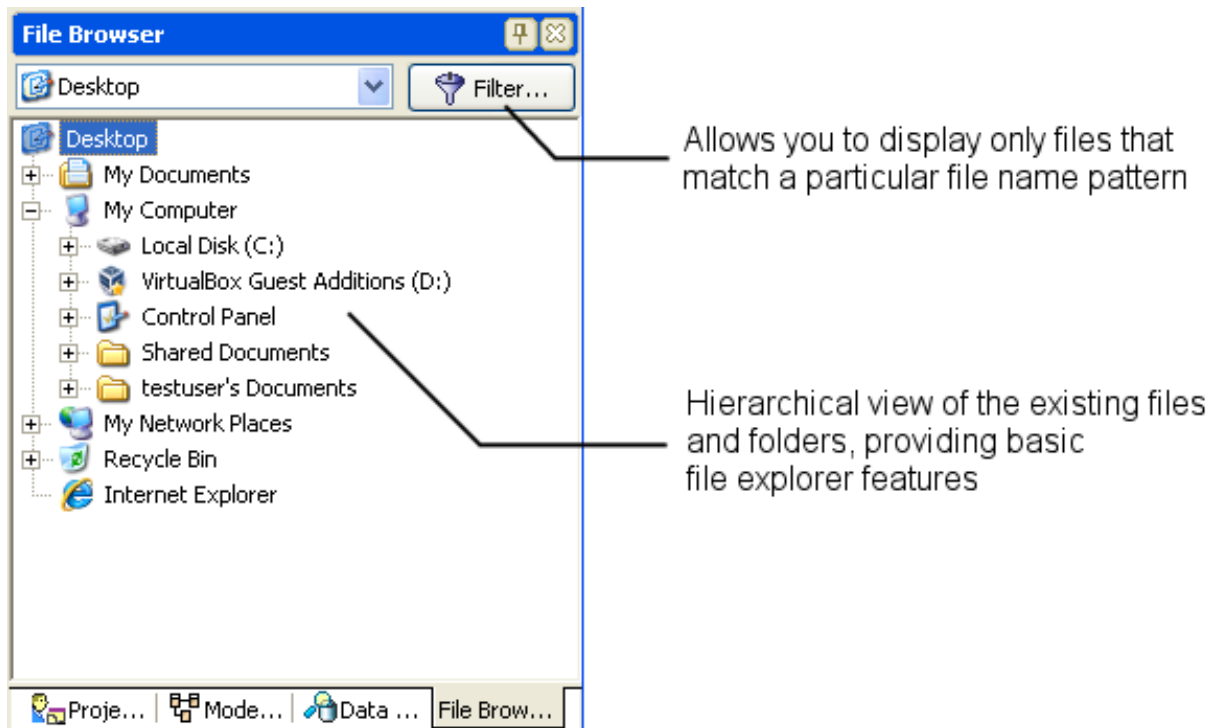


Figure 2-14. Browsing files and folders using File Browser

The **File Browser** is especially useful for managing files that are not normally part of the project itself and thus not listed in the **Project Manager**. The context menu shown by the **File Browser** when a file is right-clicked is based on the Windows Explorer menu, with two new options specific to RAD Studio. These two options allow you to either open the selected files with RAD Studio itself or to add them to the currently open project.

A useful feature of the **File Browser** is the ability to filter the displayed files based on a set of masks. After clicking the **Filter** button at the top of the **File Browser** window, a new dialog box asking for a semicolon-separated list of masks appears. For example, setting the mask to \*.txt;\*.exe displays only executables and text files.

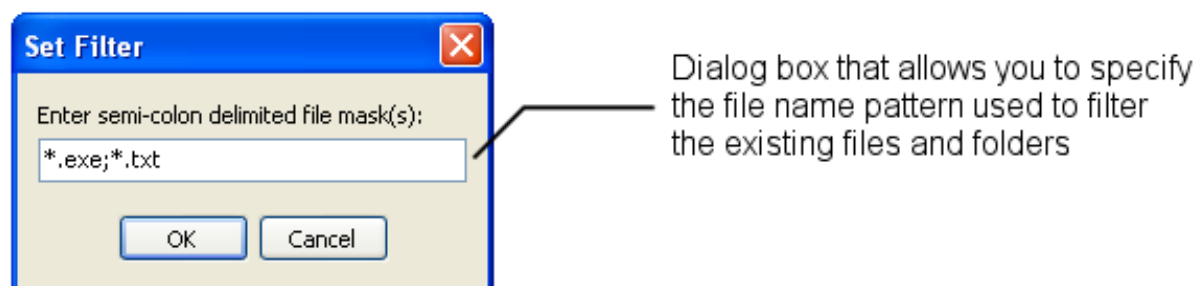


Figure 2-15. Setting up the file filter used in the File Browser

## Next

Structure View

# Structure View (IDE Tutorial)

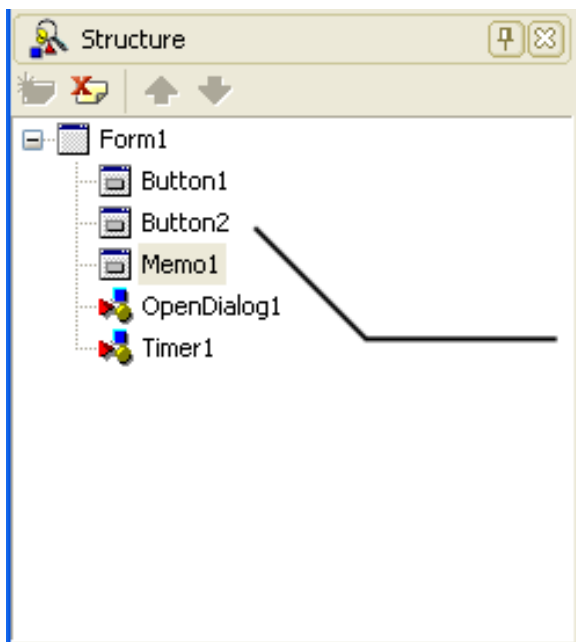
---

*Go Up to Tools Index (IDE Tutorial)*

The contents of the *Structure View* reflect the IDE's current mode. The **Structure View** shows either the hierarchy of the source code that is currently open in the **Code Editor** or the components currently displayed in the Designer. The tree diagram is synchronized with the **Object Inspector** and the **Form Designer**, so that when you change mode in the **Structure View**, the mode also changes for both the **Object Inspector** and the **Form Designer**.

To display the **Structure View**, choose **View > Structure**.

If the **Structure View** is displaying the structure of Designer components, you can single-click a component in the tree diagram to select it on the form. You can also double-click any component in the **Structure View** to open the **Code Editor** to a place where you can write an event handler for that component.



Hierarchical view of the components and source code used in the current project

*Figure 2-16. Structure View in Form Designer mode*

If displaying the structure of source code or HTML, you can double-click an item in the list to jump to its declaration or location in the **Code Editor**.

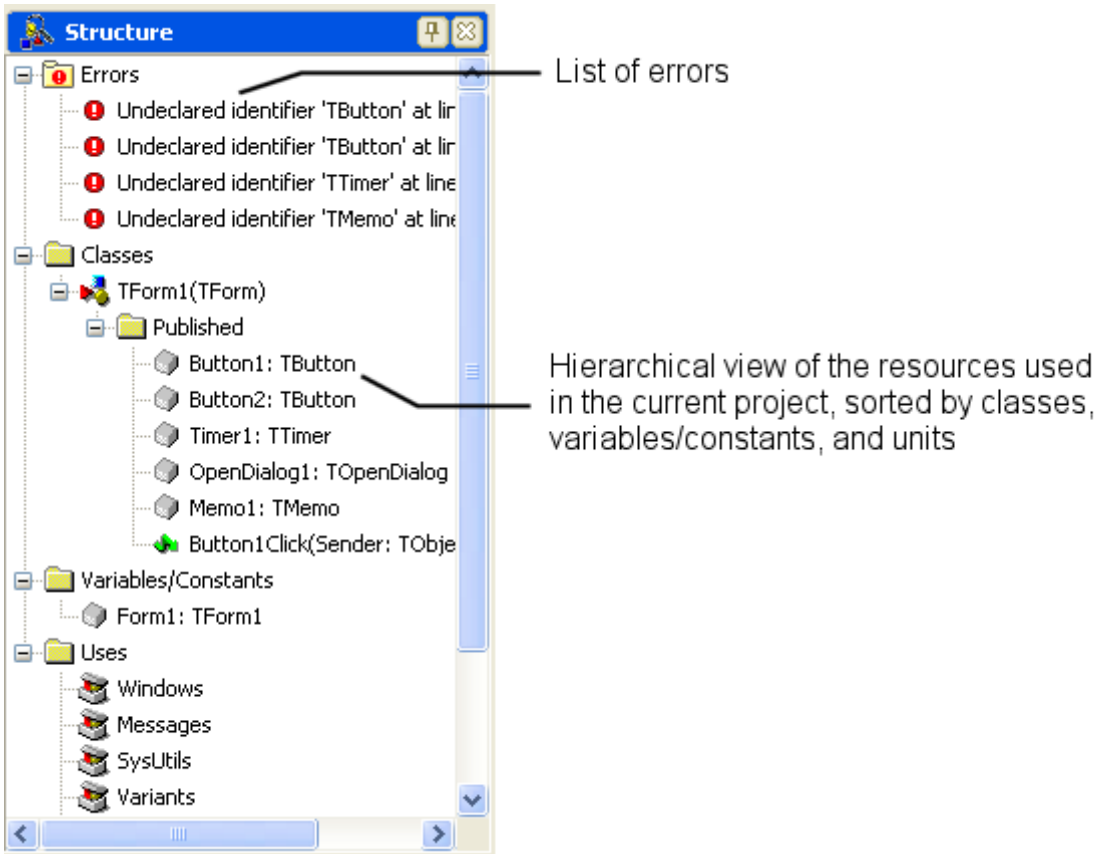


Figure 2-17. Structure View in Code Editor mode

Any code syntax errors are displayed in the **Structure View**. To locate an error in the **Code Editor**, double-click it in the **Structure View**.

You can also use the **Source View** to change components' relationships. For example, if you add panel and check box components to your form, the two components are siblings. If you drag the check box on top of the panel icon in the **Structure View**, the check box becomes a child of the panel.

The **Structure View** is also useful for displaying relationships between database objects.

You control the content and appearance of the **Structure View** by choosing **Tools > Options > Environment Options > Explorer** and changing the settings.

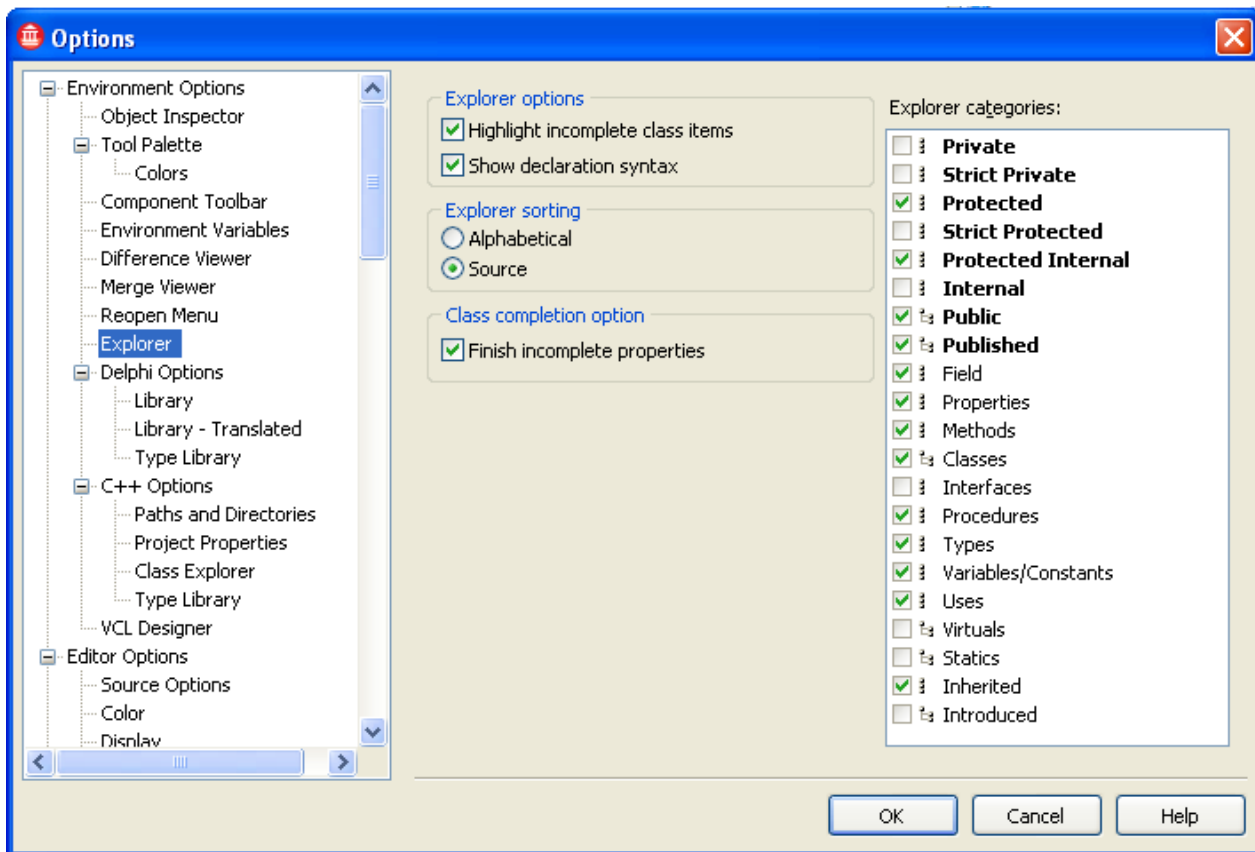


Figure 2-18. Structure View explorer options

**For more information...**

see Structure View.

## Next

The Code Editor

- Code Navigation
- Code Folding
- Change Bars
- Block Comments
- Live Templates
- SyncEdit
- Code Insight
- Refactoring
- Keystroke Macros
- To-Do Lists



# The Code Editor Index (IDE Tutorial)

---

*Go Up to Tools Index (IDE Tutorial)*

The *Code Editor* occupies the center pane of the IDE window. The **Code Editor** is a full-featured, customizable, UTF-8 editor that provides syntax highlighting, source code browsing, multiple-undo capability, and context-sensitive Help for language elements.

As you design the user interface for your application, RAD Studio generates portions of the underlying code. Whenever you modify the properties of an object, your changes are automatically reflected in the source files.

Because all of your programs share common features, RAD Studio auto-generates code to get you started. You can think of the auto-generated code as an outline that you can use to create your program.

See Code Editor for additional details on its operation.

The following topics describe the **Code Editor**'s features to help you write code.

## Topics

- Code Navigation
  - Method Hopping, Bookmarks, Finding Classes, Finding Units
- Formatting Source Code
- Code Folding
- Change Bars
- Block Comments
- Live Templates
- SyncEdit
- Code Insight
  - Code Completion, Help Insight, Class Completion, Block Completion, Code Parameter Hints, Code Hints, Error Insight, Code Browsing
- Refactoring
- Keystroke Macros
- To-Do Lists

# Code Navigation (IDE Tutorial)

*Go Up to The Code Editor Index (IDE Tutorial)*

To easily navigate in the **Code Editor**, use one of the following methods.

## Method Hopping

You can navigate between methods using a series of editor hotkeys. You can also limit hopping to the methods of the current class by setting class lock.

For example, if class lock is enabled and you are in a method of TComponent, hopping is only available within the methods of TComponent. The keyboard shortcuts for Method Hopping are as follows.


| Keyboard shortcut    | Effect                                |
|----------------------|---------------------------------------|
| CTRL+Q followed by L | toggle class lock                     |
| CTRL+ALT+UP_ARROW    | move to the top of the current method |
| CTRL+ALT+DOWN_ARROW  | move to the next method               |
| CTRL+ALT+HOME        | move to the first method in the file  |
| CTRL+ALT+END         | move to the last method in the file   |
| CTRL+ALT+MOUSE WHEEL | scroll through methods                |

Table 2-1. Method Hopping keyboard shortcuts

## Bookmarks

*Bookmarks* provide a convenient way to navigate through long files. You can mark a location in your code with a Bookmark and jump to that location from anywhere in the file.

You can use up to ten Bookmarks, numbered 0 through 9, within a file. To toggle a Bookmark at the current line, press CTRL+SHIFT+digit.

When you set a Bookmark, a book icon  is displayed in the left gutter of the Code Editor, as in the following illustration.

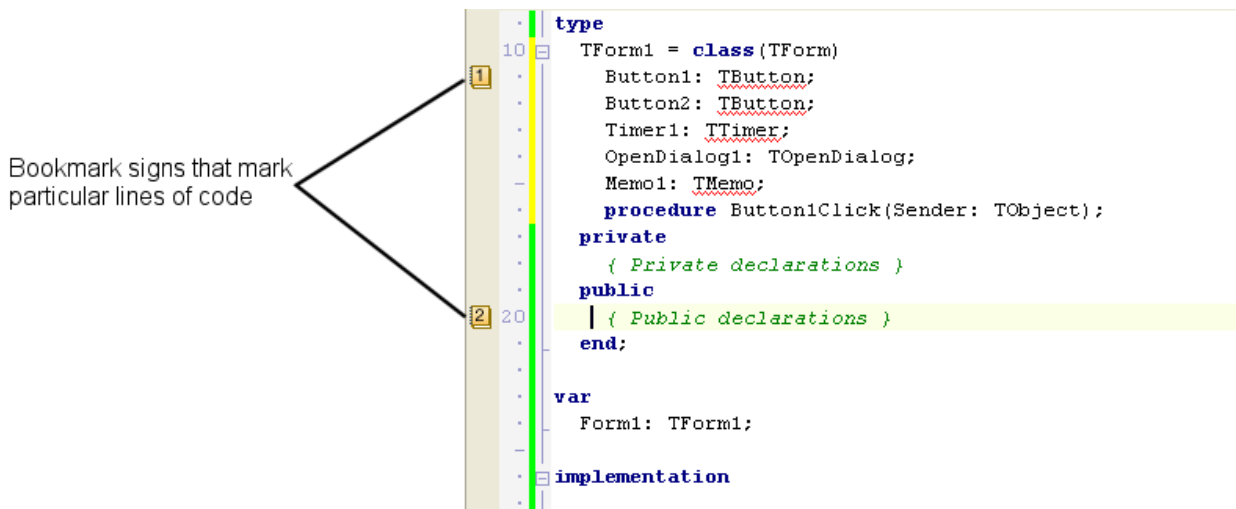


Figure 2-19. Setting Bookmarks in the source code

## Finding Classes

Use the **Search > Find Class** command to see a list of available classes that you can select. If you double-click a class, the IDE automatically navigates to its declaration.

## Finding Units

If you are using the Delphi language, you can use the **Refactor > Find Unit** command to locate and add units to your code file. As you type the class name, the list of units that match it is updated. See Find Unit for more information.

## Next

Formatting Source Code

---

# Formatting Source Code (IDE Tutorial)

---

*Go Up to The Code Editor Index (IDE Tutorial)*

For consistency and clarity, you can format source code in the **Code Editor** for both C++ and Delphi source. You can set formatting options for indentation, spaces, line breaks, alignment and capitalization.

You can format source in a couple of ways: as you are write code, or format a selection of code or the entire file using **Code Editor** context menus. You can also format a project's files.

The Options dialog (displayed by **Tools > Options**) allows you to both enable formatting and set the options for how formatting is done. C++ and Delphi each have their own formatting options. You can also create and restore profiles for different formatting styles.

See Formatting Source Code for more details on source formatting.

## Next

Code Folding

# Code Folding (IDE Tutorial)

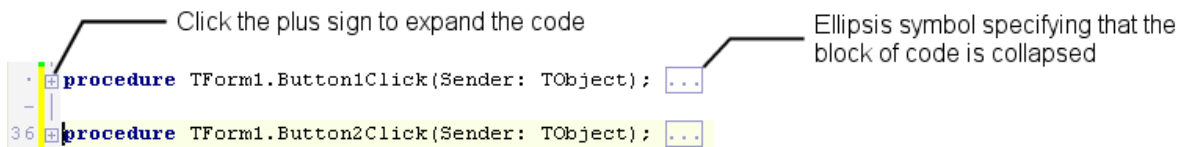
---

*Go Up to The Code Editor Index (IDE Tutorial)*

*Code Folding* lets you collapse sections of code to create a hierarchical view of your code, making it easier to read and navigate. For guidance on creating Code Folding regions, see *Using Code Folding*.

Code Folding regions have plus (+) and minus (-) signs in the gutter of the **Code Editor**, used to collapse and expand a region of code, respectively.

The collapsed code is not deleted, but hidden from view.



*Figure 2-20. Collapsed blocks of code*

## Next

Change Bars

# Change Bars (IDE Tutorial)

---

*Go Up to The Code Editor Index (IDE Tutorial)*

The left margin of the **Code Editor** displays a yellow change bar to indicate lines that have been changed but not yet saved in the current editing session. A green change bar indicates the changes that have been made since the last **File > Save** operation.

You can, however, customize the change bars to display in colors other than the default green and yellow. Select **Tools > Options > Editor Options > Color**. In the **Element** drop down menu, select **Modified Line**, then change the foreground and the background colors.

## Next

Block Comments

# Block Comments (IDE Tutorial)

---

*Go Up to The Code Editor Index (IDE Tutorial)*

You can comment-out a section of code by selecting the code in the **Code Editor** and pressing CTRL+/ (slash). Each line of the selected code is then prefixed with // and is ignored by the compiler. Pressing CTRL+/ adds or removes the slashes, based on whether the first line of the selected code is prefixed with //.

When using the Visual Studio or Visual Basic key mappings, use CTRL+K+C to add and remove comment slashes.

## Next

Live Templates

# Live Templates (IDE Tutorial)

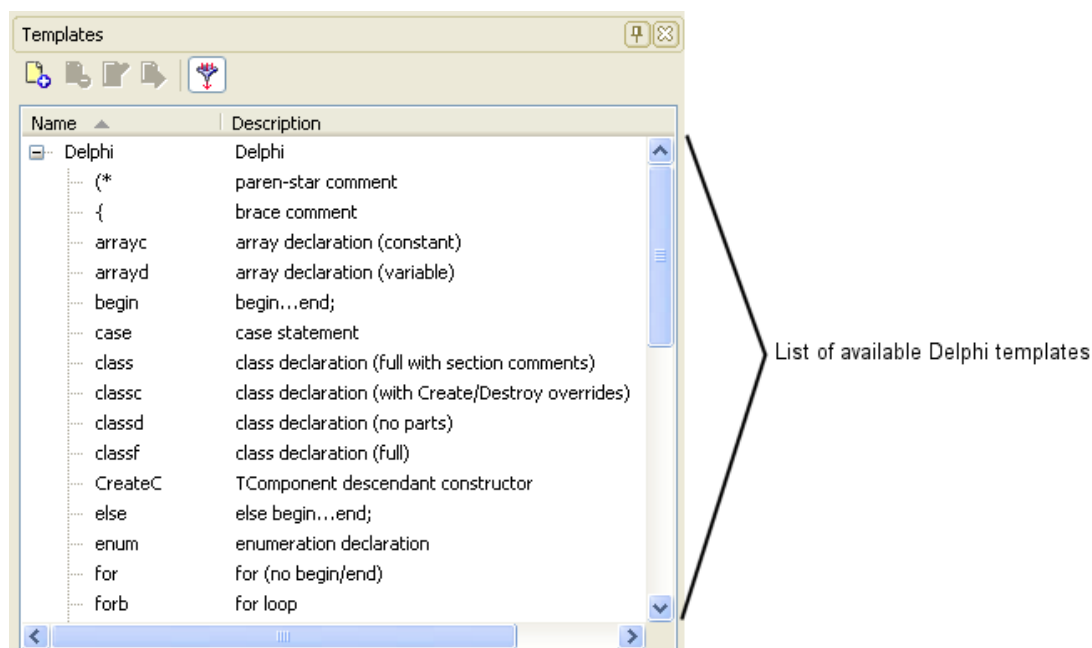
---

*Go Up to The Code Editor Index (IDE Tutorial)*

*Live Templates* allow you to have a dictionary of pre-written code that can be inserted into your programs while you are working with the **Code Editor**. You can access Live Templates by going to **View > Templates**.

Using Live Templates reduces the amount of typing that you do.

You can find further information concerning Live Templates in *Creating Live Templates* and *Using Live Templates*.



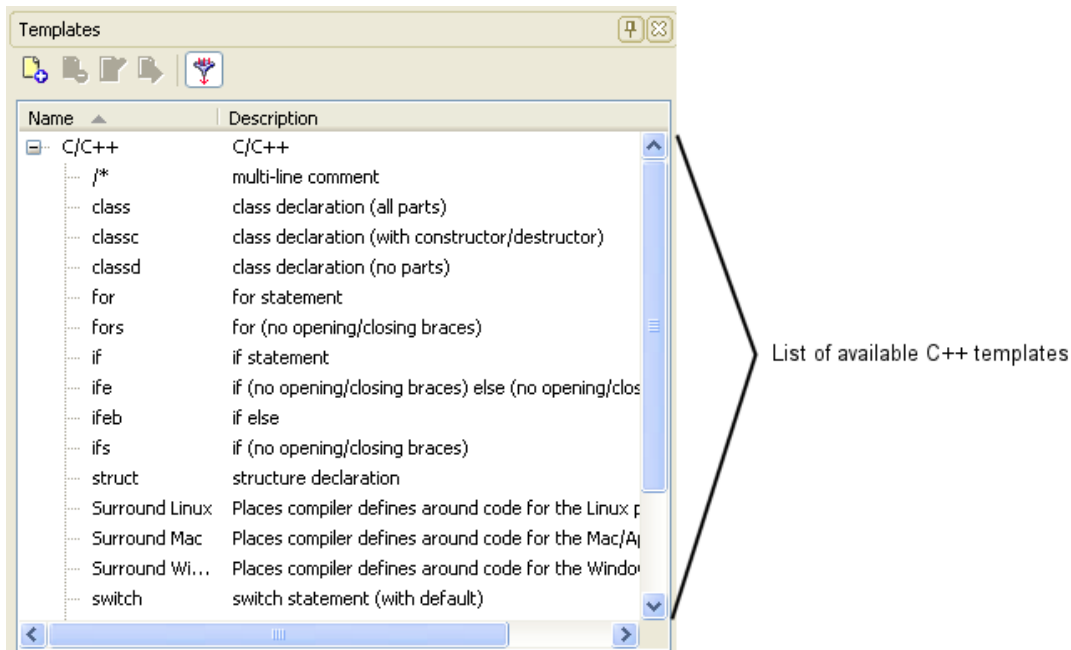


Figure 2-21. Expanding the list of Live Templates for Delphi and C++

## Next

SyncEdit

# SyncEdit (IDE Tutorial)

*Go Up to The Code Editor Index (IDE Tutorial)*

The *SyncEdit* feature lets you simultaneously edit identical identifiers in the code.

As you change the first identifier, the same change is performed automatically on the other identifiers. For example, in a procedure that contains several occurrences of the variable `TextStatus`, you can edit only the first occurrence and all the other occurrences will change automatically.

To use SyncEdit:

1. In the **Code Editor**, select a block of code that contains identical identifiers.

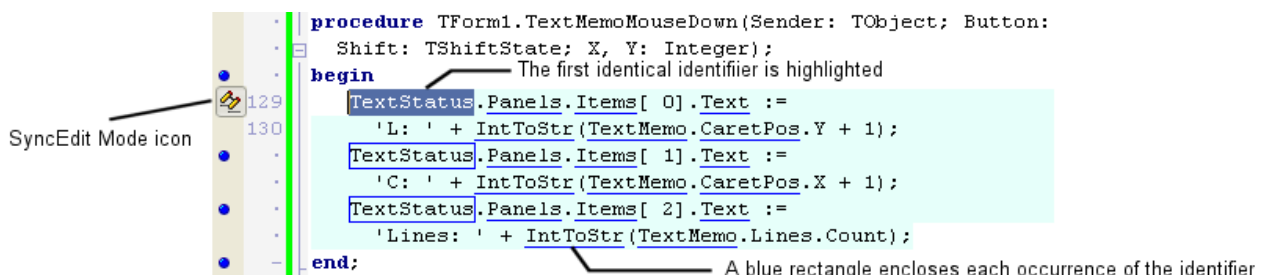



Figure 2-22. Highlighting all the occurrences of an identifier in a section of code

2. Click the SyncEdit Mode icon  that appears in the left gutter, i.e., the column to the left of the code window. The first identical identifier is highlighted and the others are outlined. The cursor is positioned on the first identifier. If the code contains multiple sets of identical identifiers, you can press TAB to move between each identifier in the selection.
3. Begin editing the first identifier. As you change the identifier, the same change is performed automatically on the other identifiers. By default, the identifier is replaced. To change the identifier without replacing it, use the arrow

keys before you begin typing.

4. When you have finished changing the identifiers, you can exit Sync Edit mode by clicking the SyncEdit Mode icon, or by pressing the Esc key.

**Note:** SyncEdit determines identical identifiers by matching text strings-it does not analyze the identifiers. For example, it does not distinguish between two like-named identifiers of different types in different scopes. Therefore, SyncEdit is intended for small sections of code, such as a single method or a page of text. For changing larger portions of code, consider using refactoring, which is a more advanced method of improving your code, without changing its functionality. Further details on refactoring are given in Refactoring.

For more information on using SyncEdit, see the video Video: Sync Editing in RAD Studio 2010, by Mike Rozlog [1].

## Next

Code Insight

## References

[1] <http://www.youtube.com/watch?v=CoeSsbJUHYY>

# Code Insight (IDE Tutorial)

---

*Go Up to The Code Editor Index (IDE Tutorial)*

*Code Insight* refers to a subset of features embedded in the **Code Editor** (such as Code Parameter Hints, Code Hints, Help Insight, Code Completion, Class Completion, Block Completion, and Code Browsing) that aid in the code writing process. These features help identify common statements you want to insert into your code, and assist you in the selection of properties and methods. Some of these features are described in more detail in the following subsections.

To enable and configure Code Insight features, choose **Tools > Options > Editor Options** and click **Code Insight**.

## Code Completion — CTRL+SPACE

To invoke Code Completion, press CTRL+SPACE while using the **Code Editor**. A popup window displays a list of symbols that are valid at the cursor location. You can type characters to match those in the selection and press Return to insert the text in the code at the cursor location.

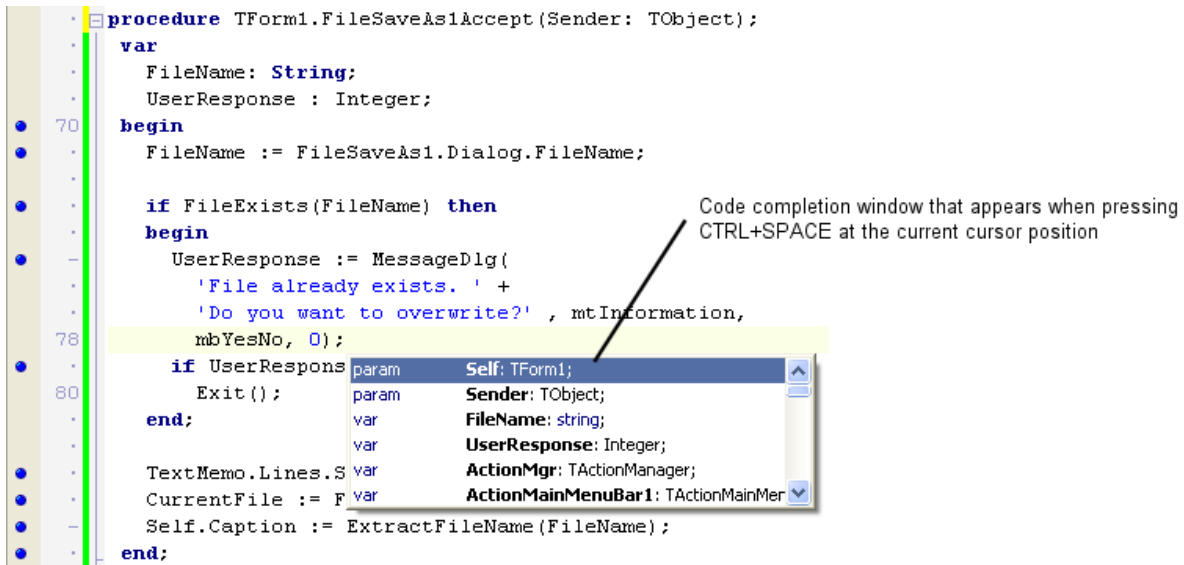


Figure 2-23. Code Completion popup window showing the list of available options

## Help Insight — CTRL+SHIFT+H

Help Insight displays a hint containing information about the symbol, such as type, file, line number where declared, and any XML documentation associated with the symbol (if available).

Invoke Help Insight by hovering the mouse over an identifier in your code, while working in the **Code Editor**. You can also invoke Help Insight by pressing CTRL+SHIFT+H.

## Class Completion — CTRL+SHIFT+C

*Class Completion* simplifies the process of defining and implementing new classes by generating skeleton code for the class members that you declare.

Position the cursor within a class declaration in the interface section of a unit and press CTRL+SHIFT+C. Any unfinished property declarations are completed.

For any methods that require an implementation, empty methods are added to the implementation section.

Class Completion can also be achieved by choosing the option **Complete class at cursor** from the Code Editor context menu.



## Block Completion — ENTER

When you press ENTER in a block of code that was incorrectly closed (while working in the **Code Editor**), a closing block token is inserted at the first empty line following the cursor position.

For instance, if you are using the **Code Editor** with the Delphi language, and you type the token begin and then press ENTER, the **Code Editor** automatically completes the statement so that you have: begin end;.

Block Completion also works for the C++ language.

## Code Parameter Hints — CTRL+SHIFT+SPACE

Code Parameter Hints display a hint containing argument names and types for method calls. You can invoke Code Parameter Hints by pressing CTRL+SHIFT+SPACE, after opening a left bracket of a function call.

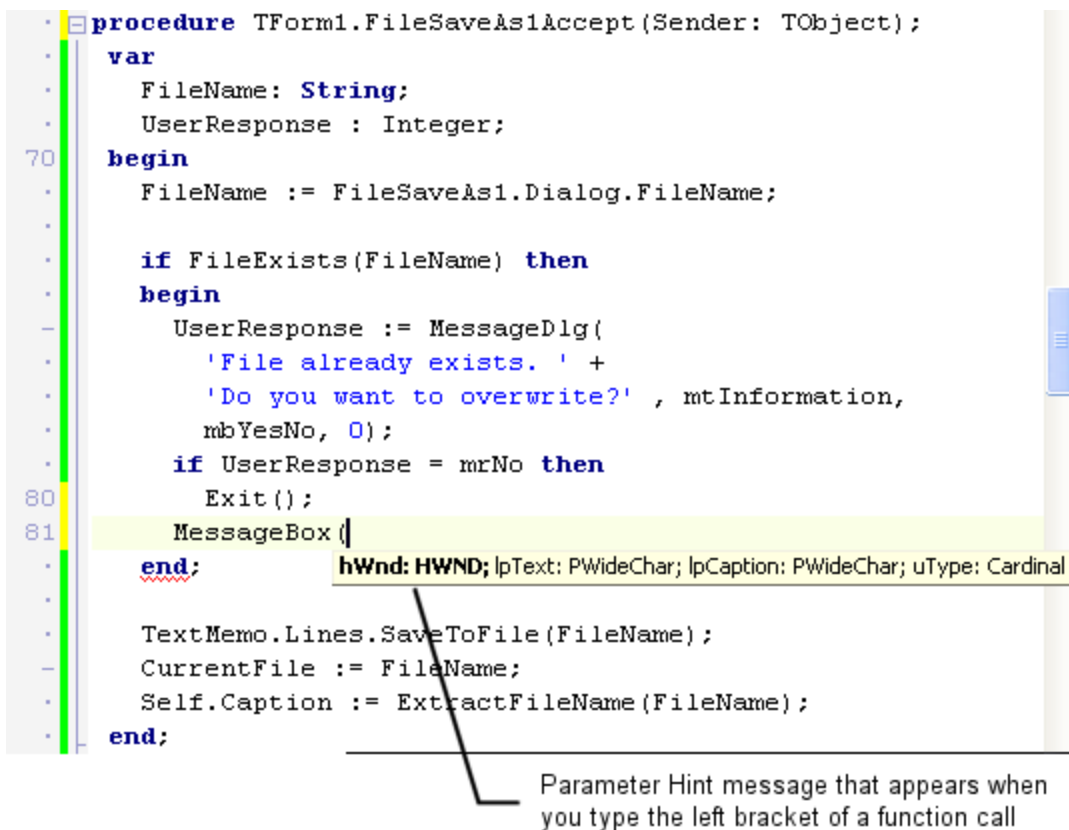


Figure 2-24. Using Code Parameter Hints to show the required types for the parameters

## Code Hints

Code Hints display a hint containing information about the symbol such as type, file, and line number, where declared. You can display Code Hints by hovering the mouse over an identifier in your code, while working in the **Code Editor**.

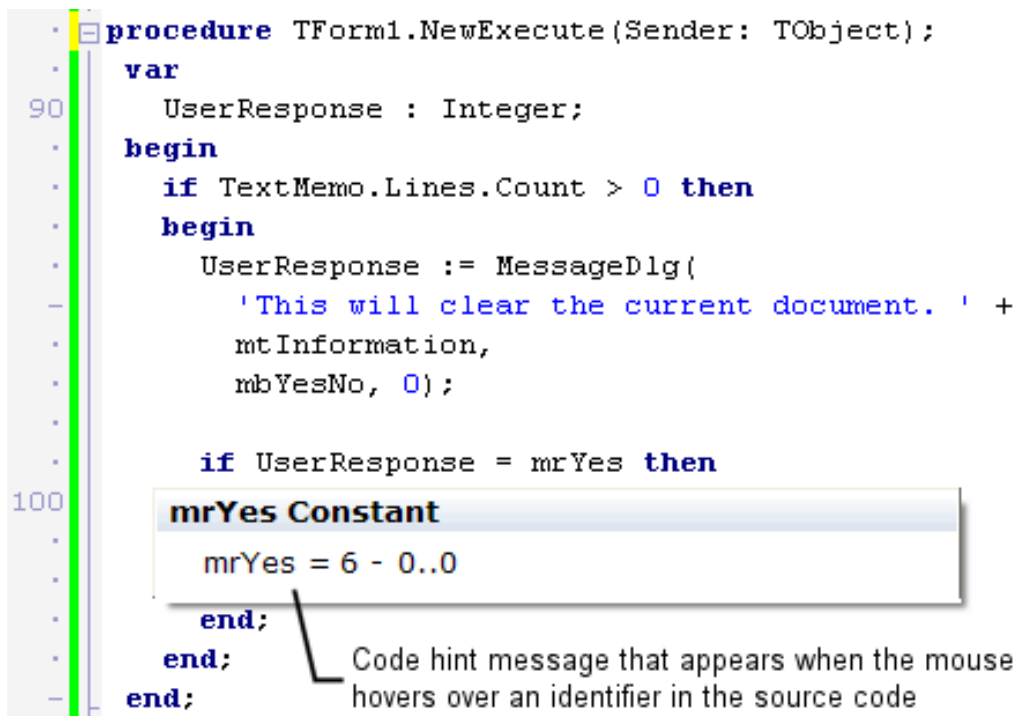


Figure 2-25. Displaying in-place Code Hints

**Note:** Code Hints only work for Delphi when you have disabled the Help Insight feature. To disable Help Insight, uncheck **Tooltip help insight** on the **Tools > Options > Editor Options > Code Insight** dialog box.

## Error Insight

When you type an expression that generates compiler errors, the expression is underlined in red.

Also, the list of errors generated by the expression appears in the Errors pane of the **Structure View**.

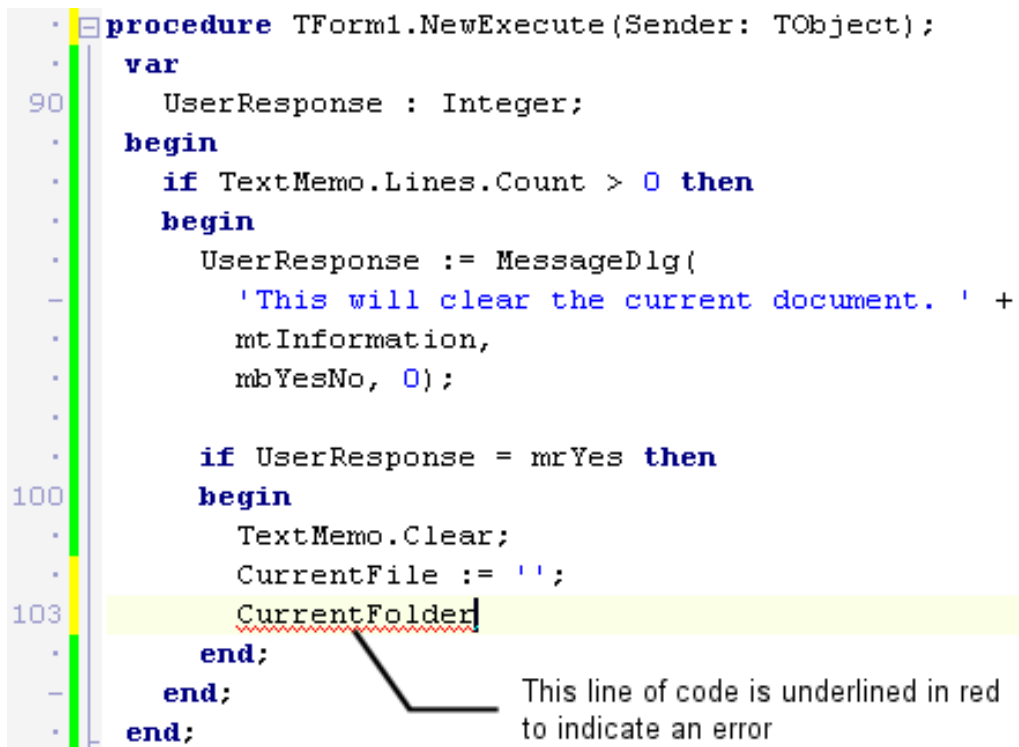


Figure 2-26. Automatic marking of errors in the code

## Code Browsing — CTRL+Click

While using the **Code Editor** to edit a VCL Form application, you can hold down the CTRL key while hovering the mouse over the name of any class, variable, property, method, or other identifier.

The mouse pointer turns into a hand and the identifier appears highlighted and underlined. Click the identifier and the **Code Editor** jumps to the declaration of the identifier, opening the source file, if necessary. You can do the same thing by right-clicking an identifier and choosing **Find Declaration**. Pressing Alt+Left Arrow returns you to where you browsed from.

*Code browsing* can find and open only units in the project Search path or Source path, or in the product Browsing or Library path. Directories are searched in the following order:

1. The project Search path
2. The project Source path, the directory in which the project was saved
3. The global Browsing path
4. The global Library path
5. The Library path, that is searched only if there is no project open in the IDE

These paths can be modified by editing the corresponding values in the list of Directories.

1. Either the project-specific **Search path** for Delphi (**Project > Options > Delphi Compiler**) or the **Include path** for C++ (**Project > Options > Directories and Conditionals**).
2. The global **Browsing path** (for Delphi: **Tools > Options > Environment Options > Delphi Options > Library** , or for C++: **Tools > Options > Environment Options > C++ Options > Paths and Directories**).

**Next**

Refactoring

## Refactoring (IDE Tutorial)

---

*Go Up to The Code Editor Index (IDE Tutorial)*

*Refactoring* is the process of improving your code without changing its external functionality.

For example, you can turn a selected code fragment into a method by using the extract refactoring method. The IDE moves the extracted code outside of the current method, determines the needed parameters, generates local variables if necessary, determines the return type, and replaces the code fragment with a call to the new method.

Several other refactoring methods, such as renaming a symbol and declaring a variable, are also available. See [Refactoring Overview](#) for more information.

**Next**

Keystroke Macros

## Keystroke Macros (IDE Tutorial)

---

*Go Up to The Code Editor Index (IDE Tutorial)*

You can record a series of keystrokes as a macro while editing code. The red button at the bottom of the code window starts the recording; the green button stops recording. After you record a macro, you can play it back to repeat the keystrokes during the current IDE session. Recording a macro replaces the previously recorded macro.

Look at [Recording a Keystroke Macro](#) for more information.

**Next**

To-Do Lists

# To-Do Lists (IDE Tutorial)

---

*Go Up to The Code Editor Index (IDE Tutorial)*

A *To-Do List* records the tasks that need to be completed for a project. After you add a task to the To-Do List, you can edit the task, add it to your code as a comment, indicate that it has been completed, and then remove it from the list.

You can filter the list to display only the tasks that are of interest to you.

To display the To-Do List window, select **View > To-Do List**. Right-click in the To-Do List dialog box and choose **Add** to display a dialog in which you can enter information on the task.

For more information on the To-Do List, see [To-Do List](#).

## Next

[History Manager](#)

# History Manager (IDE Tutorial)

---

*Go Up to Tour of the IDE Index (IDE Tutorial)*

The *History Manager* lets you compare versions of a file, including multiple backup versions, saved local changes, and the buffer of unsaved changes for the active file.

If the current file is under version control, all types of revisions are available in the **History Manager**.

The **History Manager** is displayed on the **History** tab, which lies at the center of the IDE, to the right of the **Code** tab.

The **History Manager** contains the following tabbed pages:

| Page        | Description   |
|-------------|---|
| Contents    | Displays the current and previous versions of the file.             |
| Information | Displays all labels and comments for the active file.               |
| Differences | Displays the differences between the selected versions of the file. |

*Table 2-2. History Manager pages*

The following figure shows the **Differences** page of the **History Manager**, comparing two different versions of a source file. The differences are shown using plus/minus signs, and the corresponding lines are highlighted in contrasting colors.

*Figure 2-27. Comparing two versions of a file using the Differences tab*

Revision icons are used to represent file versions in the revision lists, as described in the following table.






| Icon  | Description   |
|---|---|
|  | The latest saved file version                                       |
|  | A backup file version   |
|  | The file version that is in the buffer and includes unsaved changes |
|  | A file version stored in a version control repository               |
|  | A file version checked out from a version control repository        |

Table 2-3. Revision icons on the Differences page

## Next

Subversion Integration in the IDE

# Subversion Integration in the IDE (IDE Tutorial)

---

*Go Up to Tour of the IDE Index (IDE Tutorial)*

The well known version control package Subversion is integrated into the IDE. If you are already working with a Subversion repository, the IDE detects this fact, and the Subversion information for your files is automatically read and then displayed in the History Manager.

You can use Subversion in several ways:

- Use the **History Manager** to display Subversion history information, as well as local file history information. For example, you can diff or compare files, examine log comments, set an alternate difference viewer, and set a merge tool.
- Use the **Project Manager** to perform many common Subversion operations. The workflow with Subversion is to update your local working copy of a file, edit and save the file, and then commit your local changes to the repository.

Refer to Subversion Integration in the IDE and How To Use Subversion in the IDE for more details.

## Next

Data Explorer

# Data Explorer (IDE Tutorial)

*Go Up to Tour of the IDE Index (IDE Tutorial)*

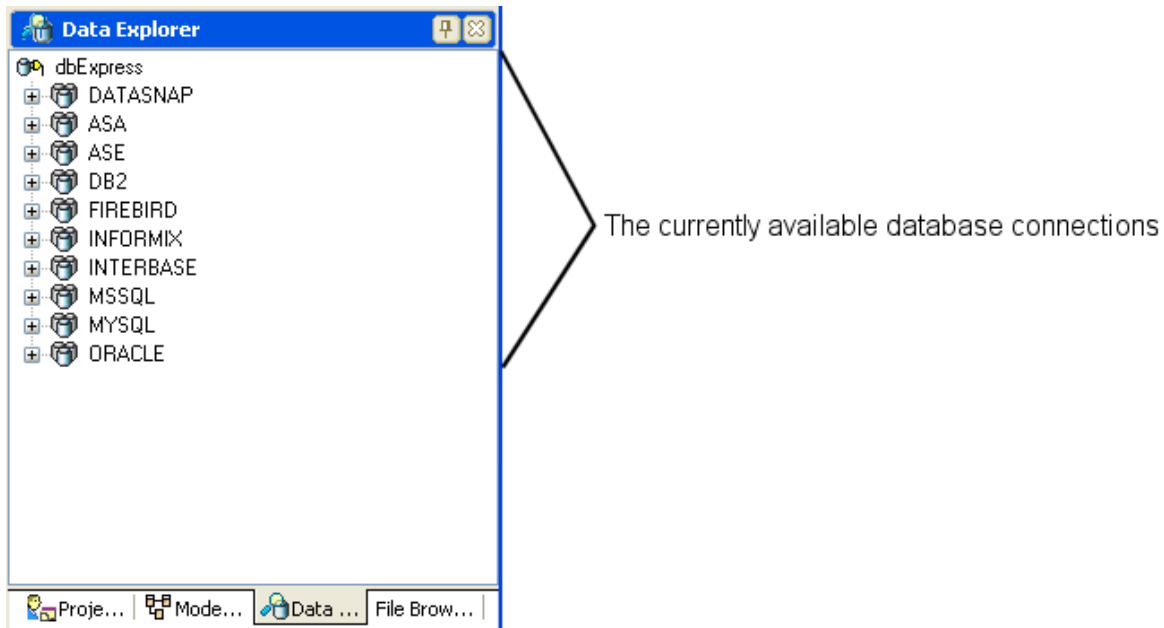


Figure 2-28. Exploring the list of available database connections

RAD Studio offers a variety of database and connectivity tools to simplify the development of database applications.

The *Data Explorer* is located, by default, in the upper right corner of the IDE. The **Data Explorer** allows you to create and modify database connections that can easily be used later in your database applications.

**Note:** **Data Explorer** works for databases that use the dbExpress connection type.

After you create a database connection, you can use the **Data Explorer** to create, view and modify tables, views, procedures, function, and synonyms. You can click an item from the expanded connection type entry. A menu that allows you to refresh the data or create a new item will appear.

For more information, see *Data Explorer*.

## Next

Object Repository

---

# Object Repository (IDE Tutorial)

---

*Go Up to Tour of the IDE Index (IDE Tutorial)*

The *Object Repository* (**Tools > Repository**) allows you to share forms, dialog boxes, frames, and data modules. It also provides templates for new projects and wizards that guide the user through the creation of forms and projects.

For more information, see [Using the Object Repository](#).

## Next

[IDE Insight](#)

---

# IDE Insight (IDE Tutorial)

---

*Go Up to Tour of the IDE Index (IDE Tutorial)*

*IDE Insight* allows you to search through and immediately use all of the IDE's various elements. The IDE Insight dialog (displayed by **Search > IDE Insight** or CTRL+.) allows you to search in two ways:

- You can look through an expandable list of categories of IDE elements. The categories available depend on whether you are using the **Form Designer** or the **Code Editor**.
- You can type a string, and the IDE Insight dialog displays all the items in each category that match any part of the string. The list is updated as you type.

The IDE categories available are Code templates, menu commands, **Tool Palette** components, Desktop SpeedSettings, source files in open projects, forms and data modules, new items from **File > New > Other**, **Object Inspector**, open files, preferences from **Tools > Options**, project options, recent files, and recent projects.

To use an item found by IDE Insight, double-click the item. The action taken depends on the item. Double-clicking a menu item performs the menu item. For example, double-clicking preferences displays the appropriate preference dialog. Double clicking an **Object Inspector** item highlights that property in the **Object Inspector**.

IDE Insight puts the facilities of the IDE in a categorized list for simple access.

See [IDE Insight](#) for more details.

## Next

[Starting your first RAD Studio application](#)



# Starting your first RAD Studio application Index (IDE Tutorial)

---

*Go Up to Tutorial: Using the IDE for Delphi and C++Builder*

This section explains how to use the Rapid Application Development tools of Embarcadero RAD Studio to create a GUI (Graphical User Interface) application. You start with creating the main form, customizing it, and adding the necessary visual and nonvisual components.

The sections of code in the application to handle user actions are called *event handlers*, which you also need to implement. User events handled include clicking items under the menus and typing or clicking inside the edit window.

After following the steps, given both for Delphi and C++, you will have a basic text editor, with a few additional features like word-wrapping and the ability to change the font and display or the current cursor position in the status bar.

## Topics

- Using project templates from the Object Repository
- Basic customization of the main form
- Adding the components using the Form Designer
- Customizing the components
- Coding responses to user actions in the Code Editor
- Compiling and running the application
- Debugging the application

# Using project templates from the Object Repository (IDE Tutorial)

Go Up to *Starting your first RAD Studio application Index (IDE Tutorial)*

In the IDE, you typically create projects by opening project templates from the Object Repository, and you do this by clicking **File > New**.

To create this project in RAD Studio, click **File > New** and then (depending on the language you want to use) choose either:

- **VCL Forms Application > C++Builder**
- **VCL Forms Application > Delphi.**

At this point, the **File** menu and its **New** command submenu should be displayed as in the two following screenshots.

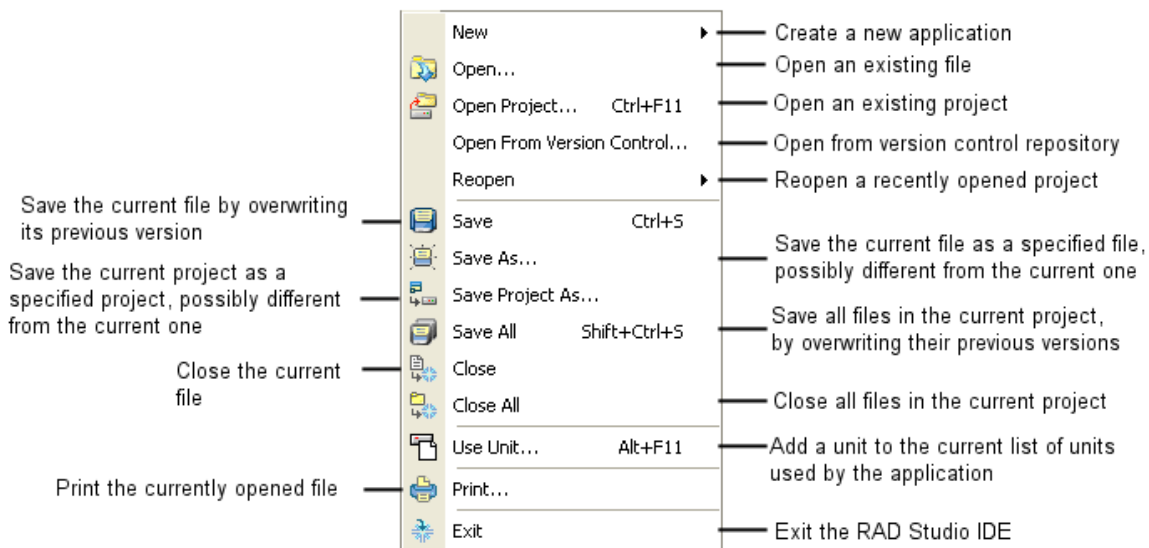


Figure 3-1. Description of basic options in the File menu

Figure 3-2. The File > New submenu

This tutorial uses the **VCL Forms Application** template available on the **File > New** submenu. But many additional project templates are available on the **New Items** dialog box, which you can open by choosing **File > New > Other....**

Figure 3-3. The New Items dialog box

From the **New Items** dialog box, you can create many types of C++ or Delphi projects, as well as unit tests, Web documents, 3rd-party projects, and more.

After you click the **VCL Forms Application** menu item to create a VCL forms project, several files are automatically generated, including the main form. After files are generated, the main form is displayed in the **Form Designer**. The next figures show screenshots of the IDE at this step, for Delphi and C++Builder, respectively.

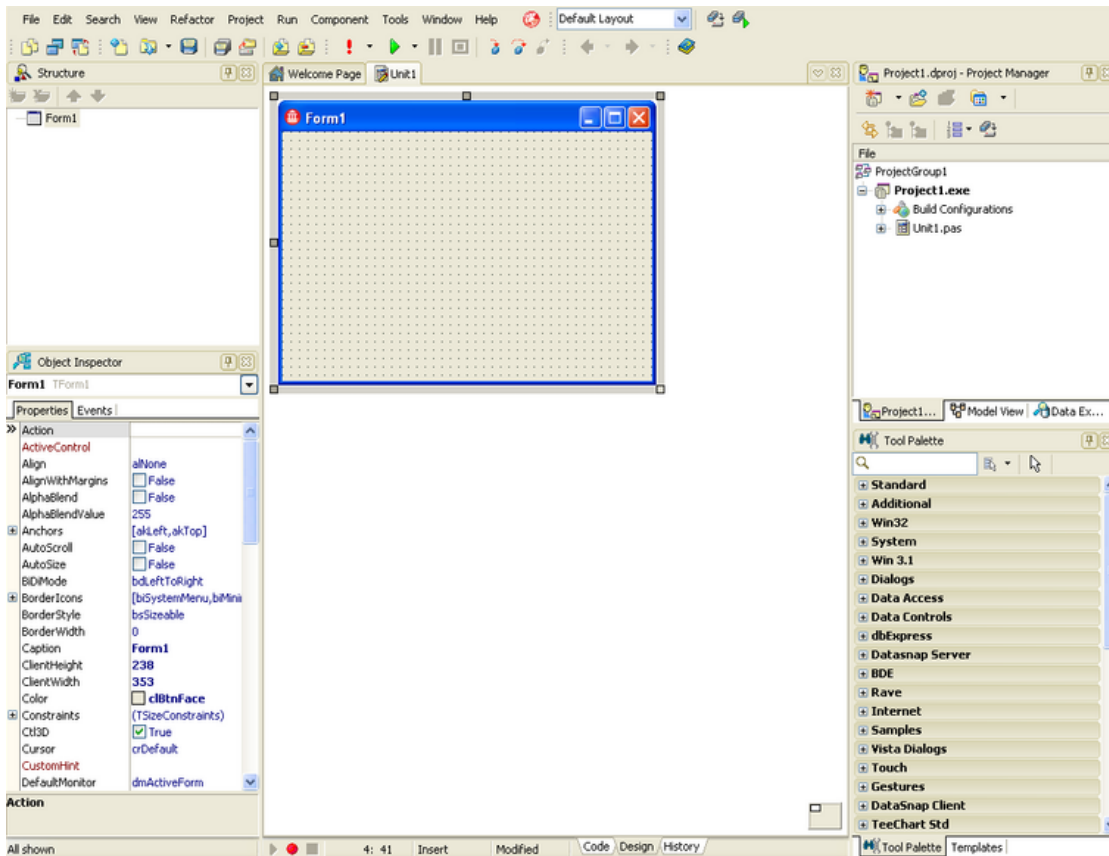


Figure 3-4. The default layout for creating a RAD Studio application (Delphi view)

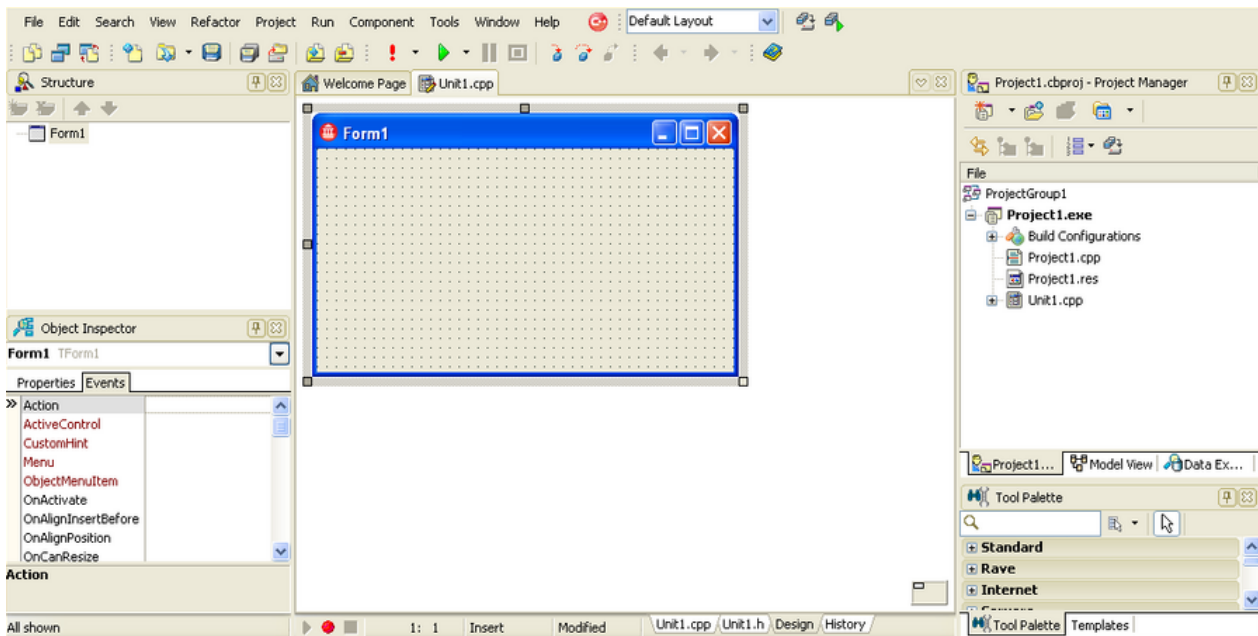


Figure 3-4a. The default layout for creating a RAD Studio application (C++ Builder view)

## Next

Basic customization of the main form

# Basic customization of the main form (IDE Tutorial)

*Go Up to Starting your first RAD Studio application Index (IDE Tutorial)*

Before adding any components, you should start by doing some basic customization to the form. Make sure that the main form is activated (click it once to activate) and that the **Object Inspector** window is visible in the lower left quadrant (if not, press F11 to display it). With the **Properties** tab selected, look for the *Caption* property and change its value to Text Editor; also, change the value of the *Name* property to TextEditorForm.

To make the design of the project more visually balanced, set the main form to initially be positioned in the center of the screen. To do this, change the value of the *Position* property to `poScreenCenter` by clicking the value field for *Position* and then selecting the value from the dropdown list. For the same reason, make the form square-shaped, by changing the values of both *Width* and *Height* to 400, or any other number you prefer, as long as the number does not exceed the current screen size.

After making these changes, the main form should look like the following figures, using Delphi or C++Builder, respectively.

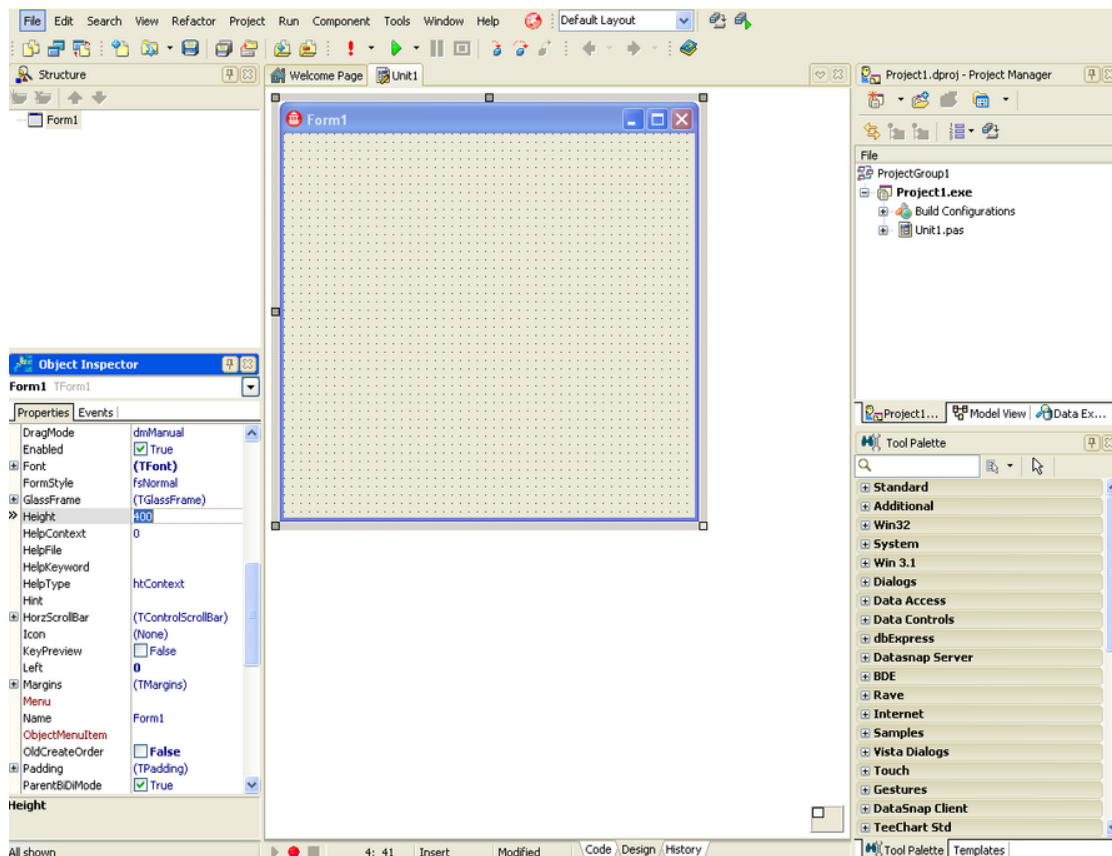


Figure 3-5. Basic customization of the main form (Delphi view)

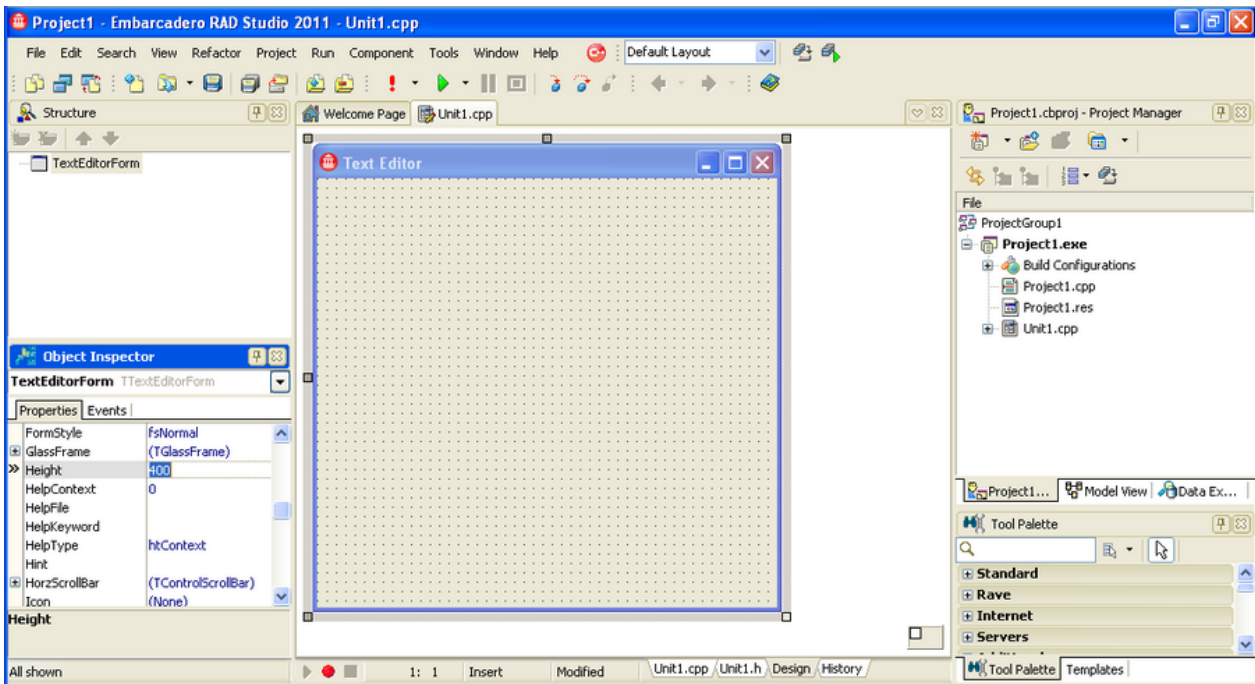


Figure 3-6. Basic customization of the main form (C++Builder view)

## Next

Adding the components using the Form Designer

# Adding the components using the Form Designer (IDE Tutorial)

*Go Up to Starting your first RAD Studio application Index (IDE Tutorial)*

Now that you have set up the main form, you can proceed with arranging the necessary components to create your text editor application. First, you need to add a menu bar providing the basic options for file manipulation, editing, and other options like changing the font or toggling word wrap.

## Adding an action manager

Add an action manager to the form to automatically provide the basic functionality of your application. To do so, make sure the **Design** tab is selected, go to the **Tool Palette** and type "action" in the search box. This results in displaying only components with the string "action" in their name, including the **TActionManager** component. The **Tool Palette** should look like the following image.

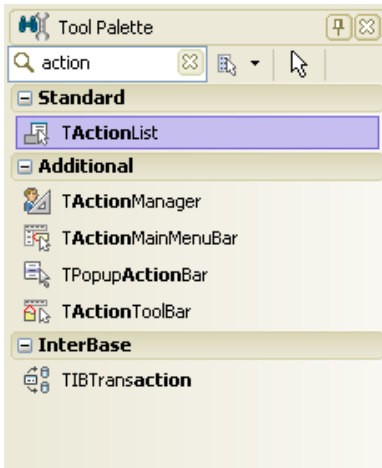



Figure 3-7. Using the action filter in the **Tool Palette** to select **TActionManager**

Double-click the **TActionManager** button to add it to the form. Now you should change the name of the **ActionManager** to suit your application. To do this, make sure **Object Inspector** is visible (if not, press F11 to display it) and click the **Action Manager** icon  to activate it. In the **Object Inspector**, click the *Name* property and change its value to ActionMgr.

### Adding the main menu

To add a main menu bar to the form, locate the **TActionMainMenuBar** component in the **Tool Palette**. Double-click **TActionMainMenuBar** to add it to the form.

### Adding a Status Bar

Next, place a status bar on the form. To do this, type "status" in the search box of the **Tool Palette** to easily locate the **TStatusBar** component. Using this filter, the **Tool Palette** looks like the following screenshot.

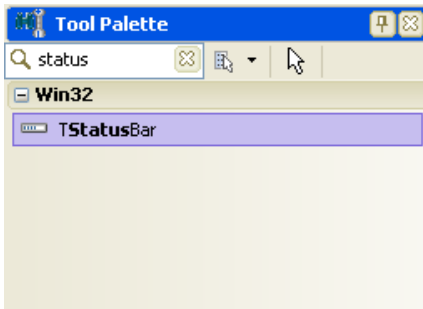


Figure 3-8. Using the status filter in the **Tool Palette** to select **TStatusBar**

As with the previous components, double-click **TStatusBar** to add it to the form.

## Adding a text box

The only component left to add is a text box, giving your application its main functionality—that of a text editor. Type "memo" in the search box and locate the **TMemo** component. The **Tool Palette** should now display the components whose names include the text "memo", as in the following image.

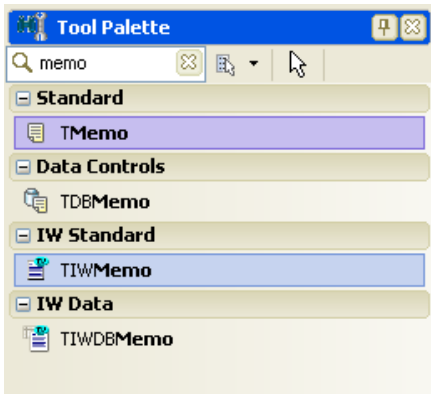


Figure 3-9. Using the memo filter in the Tool Palette to select **TMemo**

Double-click **TMemo** to add it to the form. The main form should now display the action manager, the action main menu bar, the status bar, and the memo we have previously added to the form, similar to the following image.

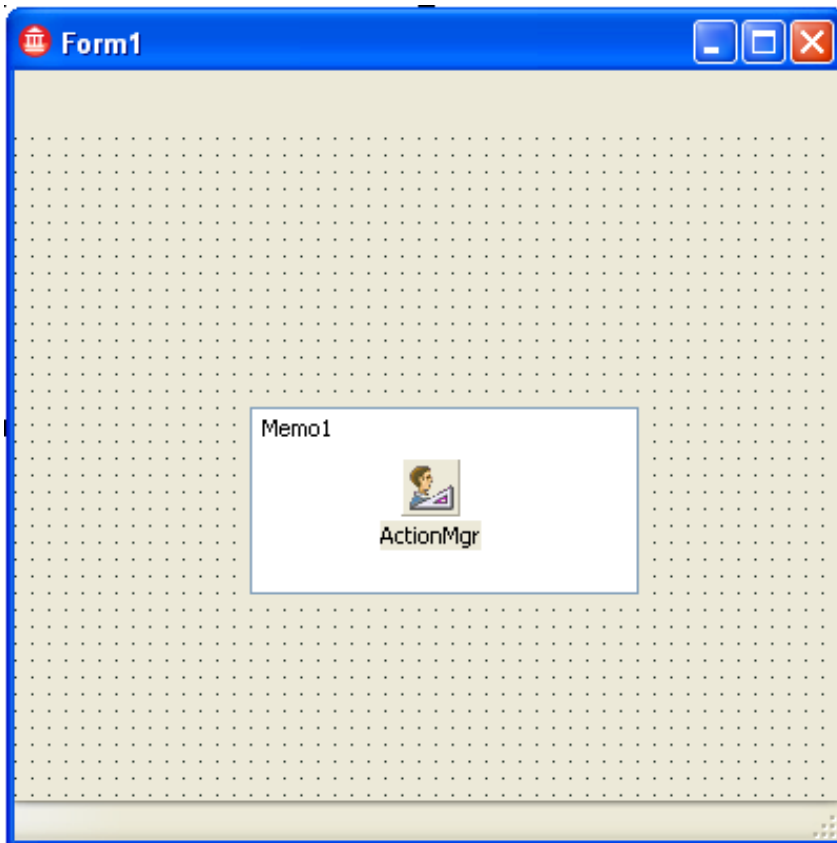


Figure 3-10. Basic text editor form

## Adding the main menu commands

To finish designing the form, you must add the items in the main menu. Start by double-clicking the action manager component on the form to open the **Actions** editor. The **Action Manager** dialog should be displayed.

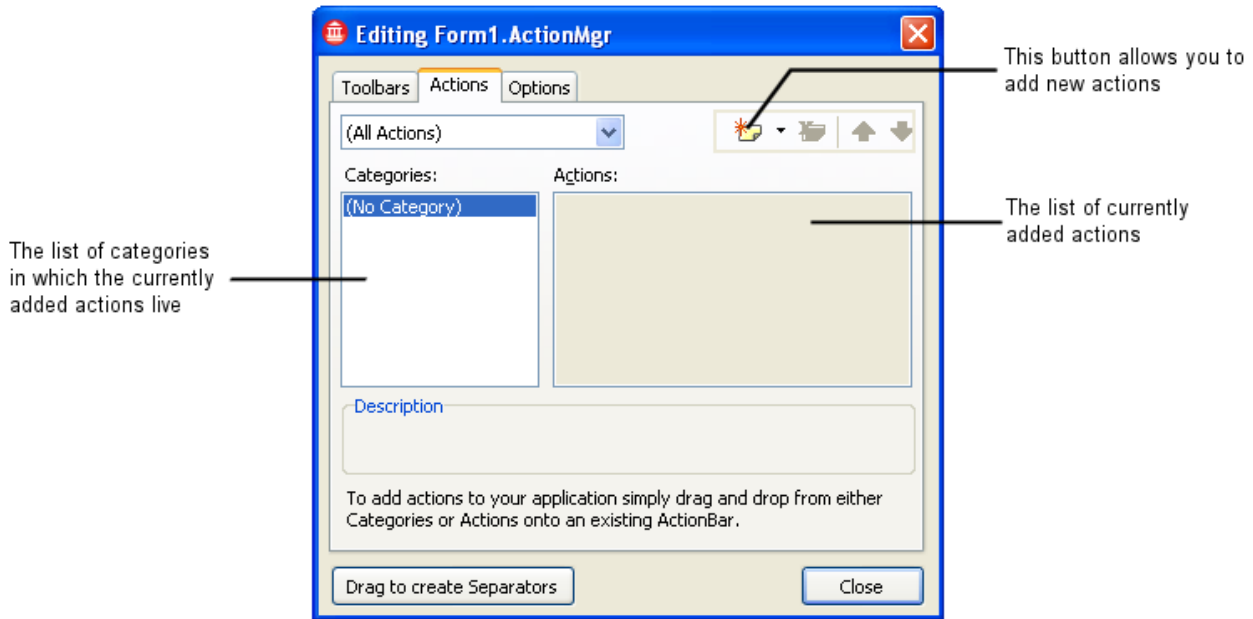


Figure 3-11. The main elements of the **Actions** page in the **Action Manager**

You are now ready to create the items in the main menu. Press CTRL+Insert to add new standard actions or click the down-arrow of the **New Action** icon and choose **New Standard Action...** from the menu. The image below shows this menu.



Figure 3-12. Adding a New Standard Action

While pressing CTRL, select all items in the **Edit** category and also **TFileOpen**, **TFileSaveAs**, and **TFileExit** from the **File** category, then click **OK**.

The following image shows how the Standard Action Classes list should display, with the items in the **File** category selected.

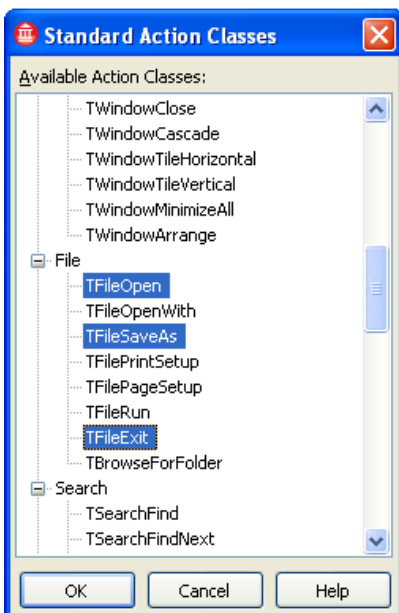


Figure 3-13. Selecting the Standard Actions that implement the basic file and text operations



After clicking **OK**, wait until the menu items are automatically generated. You may notice that the object inspector displays the properties of each menu item as it is created.

### Defining Action Properties

The text editor needs a few capabilities other than the standard actions, so you must define your own custom actions. To do this, from the **Actions Editor**, select *File* from the **Categories** list and click the **New Action** button twice to create two new, non-standard actions under the **File** menu.

You can now customize the newly created actions. Click *Action1* in the **Actions** list and use the **Object Inspector** to change its *Name* property to **New** and its *ShortCut* to CTRL+N. Click *Action2* and change its *Name* to **Save** and its *ShortCut* property to CTRL+S.

Now, use the **Move Up** and **Move Down** arrows to put the actions in the right order, as in the following image.

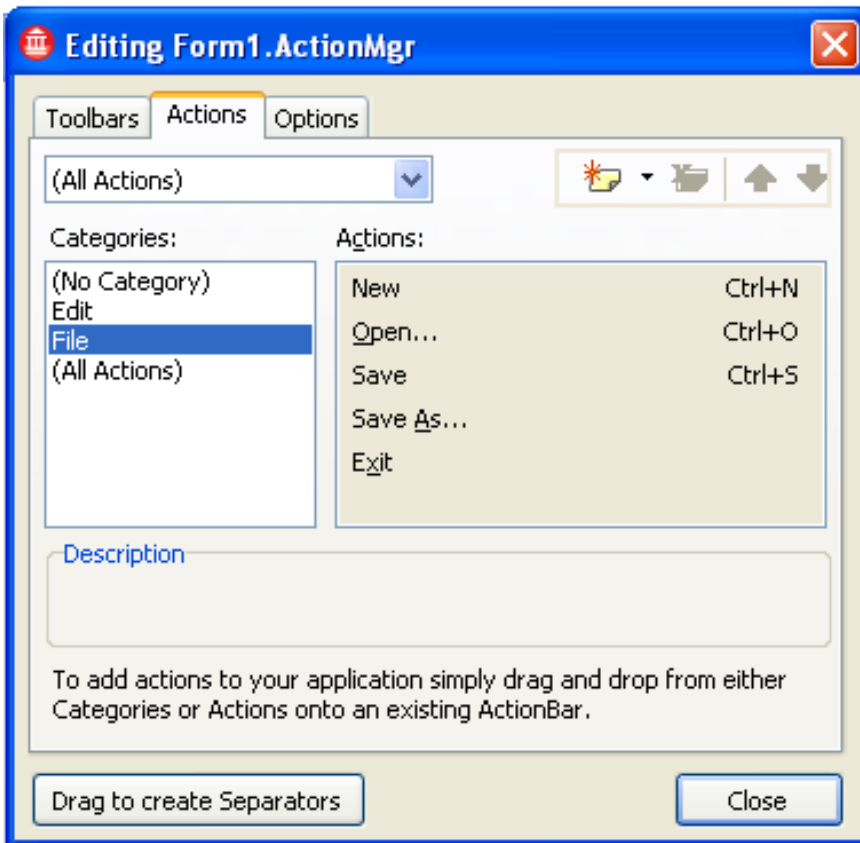


Figure 3-14. Arranging the actions in the *File* menu and finishing adding the Standard Actions

### Adding word wrap and font selection capabilities

To give your text additional features—word-wrapping and the ability to change the font—you need to add a menu to the main menubar. Click *(No Category)* from the **Categories** list and press CTRL+Insert on the keyboard to create a new standard action. The **Standard Actions Classes** list is displayed.

Select **TFontEdit** from the **Dialog** category and click **OK**. In the Actions list, click **Dialog**, then click *Select Font* and use the **Object Inspector** to change its *Category* property to **Format**. Do this by selecting *Category* and type the word **Format**. Also, enter **Font** as its *Caption* property.

With the **Format** category selected in the **Categories** list, press the **New Action** button to define a new action. Change its *Name* to **WordWrap** and its *Caption* to **Word Wrap**, using the **Object Inspector**.

Now drag each item from the **Categories** list to the menu bar at the top of the main form, in this order: **File**, **Edit**, **Format**.

The following image shows how the File menu should now look.

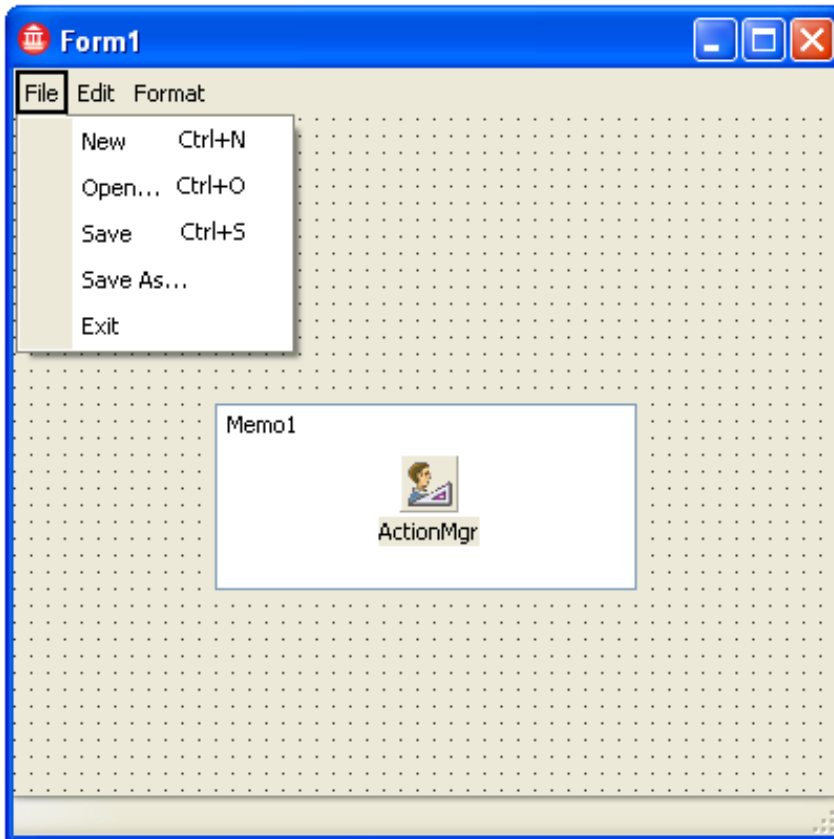


Figure 3-15. The final look of the File menu

Finally, close the **Actions Editor** to continue customizing the application.

## Next

Customizing the components

# Customizing the components (IDE Tutorial)

*Go Up to Starting your first RAD Studio application Index (IDE Tutorial)*

In the previous section, Adding the components using the Form Designer, you have added all the required components to your form and then configured the action manager. Before you continue with writing code for the event handlers, customize the properties of the newly placed components.

To customize a component, select the component in the **Form Designer**. You can then edit the properties of the selected component in the **Object Inspector**.

Follow these steps to customize the memo component:

1. Select the memo component in the **Form Designer** by clicking the memo component.
2. Find the memo component in the **Object Inspector** (if the **Object Inspector** is not visible, press F11 or click **View > Object Inspector**).
3. Set the *Align* property to `alClient`. This makes the memo component occupy all the free space available on the form.
4. Set the *Name* property to `TextMemo`. Naming your component properly is very important because your code needs to access the component using that name. Setting a name you can easily remember is useful.
5. Set the *ScrollBars* property to `ssBoth`. This results in displaying both the vertical and horizontal scroll bars in the memo and allows users to easily scroll through its contents.
6. Set 'WordWrap' to `False`. *WordWrap* tells the memo to wrap all text on several lines if the text does not fit in a single line. A `False` value disables word wrapping.
7. Find the *Lines* property and press the "..." button located in the value box. A dialog appears that allows you to edit the initial contents of the memo, as in the figure below. Delete all the text and then press **OK** to clear the memo.

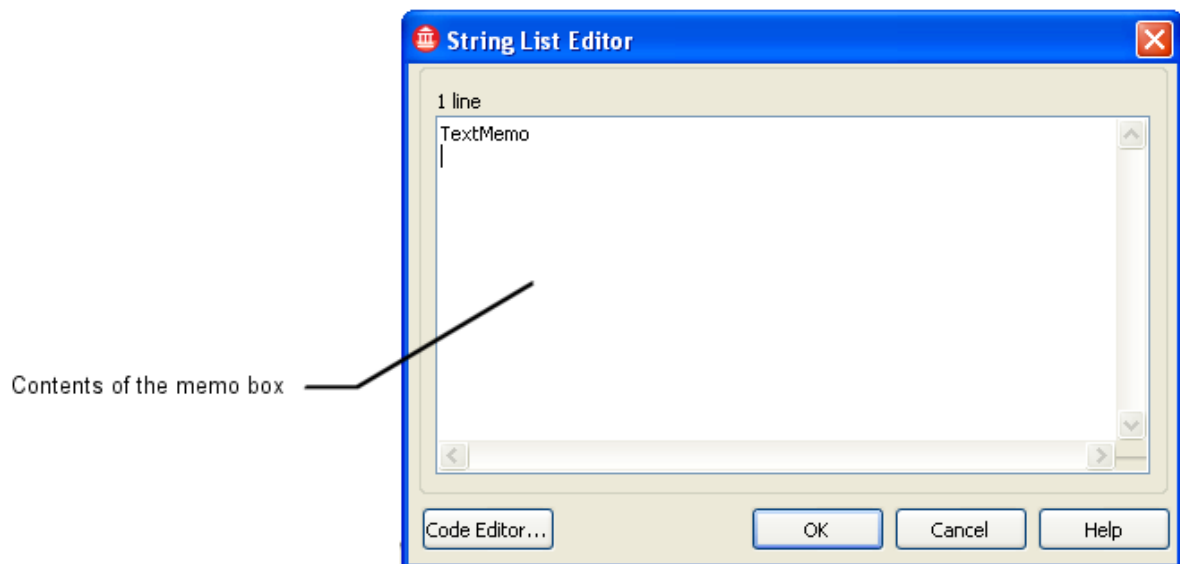


Figure 3-16. Editing the contents of the memo

After you have customized the memo, customize the status bar as follows.

1. Select the status bar in the **Design** window.
2. Set the *Name* property to `TextStatus`.
3. Find the *Panels* property and press the "..." button at the right side of the value box. This displays a new dialog box that allows adding and customizing panels displayed in the status bar.
4. Press the Insert key three times to add three panels. The panel editor should look like the following figure. You do not need to customize these panels, so just close the dialog.

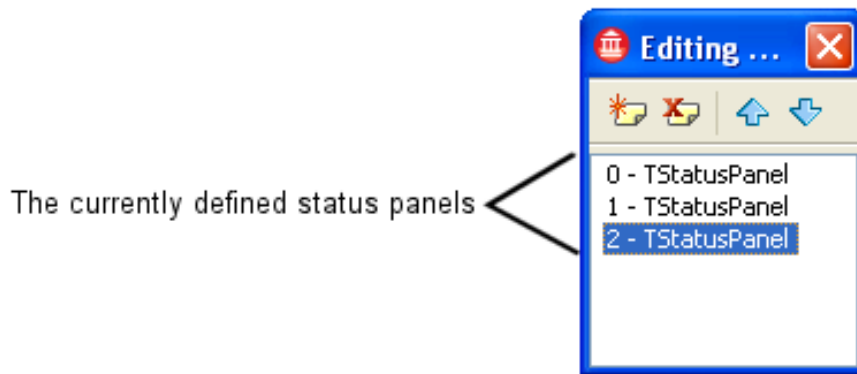


Figure 3-17. Panel editor showing the list of added status panels

This completes customizing the components.

Before writing code, save all the changes you have made to the project. Click **File > Save As** and save the unit as `TextEditor`. Also, click **File > Save Project As** and save the project as `TextEditor_proj`.

## Next

Coding responses to user actions in the Code Editor

# Coding responses to user actions in the Code Editor (IDE Tutorial)

---

*Go Up to Starting your first RAD Studio application Index (IDE Tutorial)*

By following the instructions in this section, you will make your application interactive and provide it with the functionality you want. You will code the event handlers, that is, the responses to clicking the various items in the main menu.

## Beginning the code

Begin writing code by defining a `String` variable that you use throughout the execution of the application to retain the name of the currently opened text file. Make sure you are in **Code Editor** mode by selecting the **Code** tab, next to the **Design** tab in the status bar. To toggle between **Form Designer** and **Code Editor** mode, press F12.

In Delphi, define a `String` variable called `CurrentFile` in the private section of the `TTextEditorForm` class in the interface part, as in the following figure.

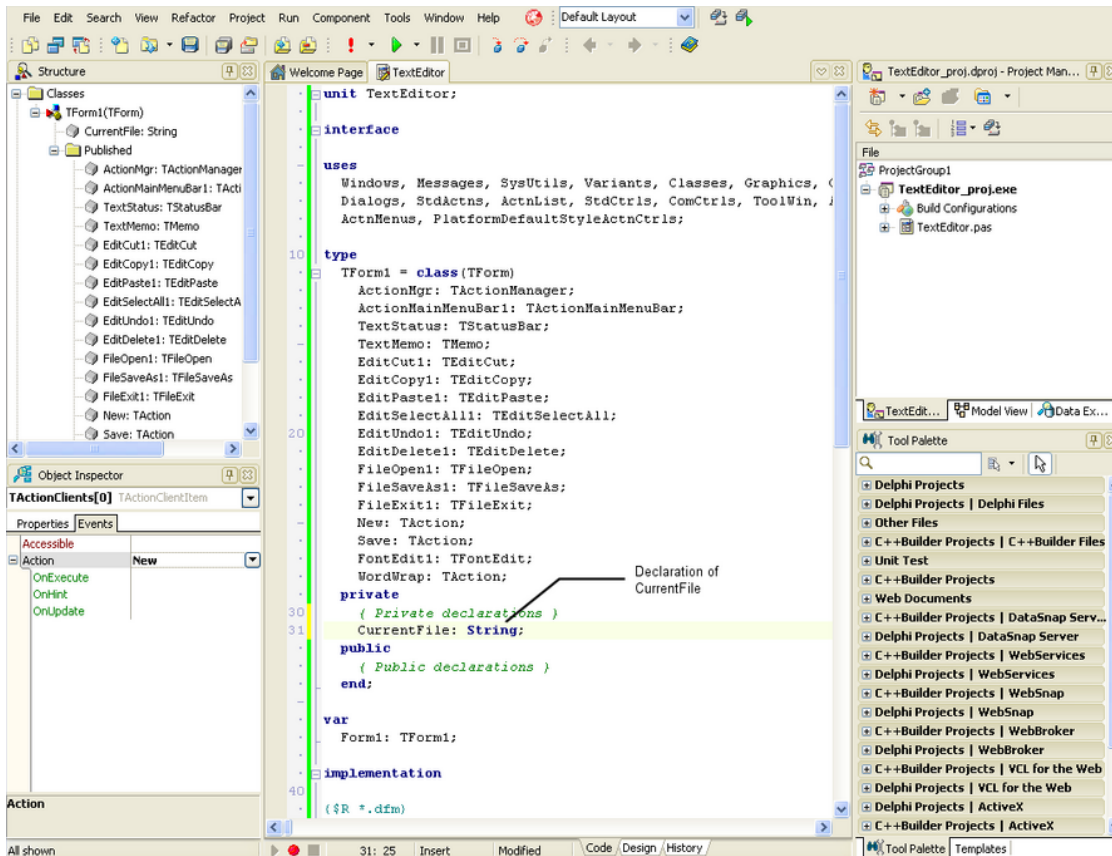


Figure 3-18. Defining the `CurrentFile` private variable (Delphi view)

In C++, use the tabs at the bottom of the **Code Editor** window to display the `TextEditor.h` file. Declare the `currentFile` variable in the private section of `TTextEditorForm`, as in the following figure.

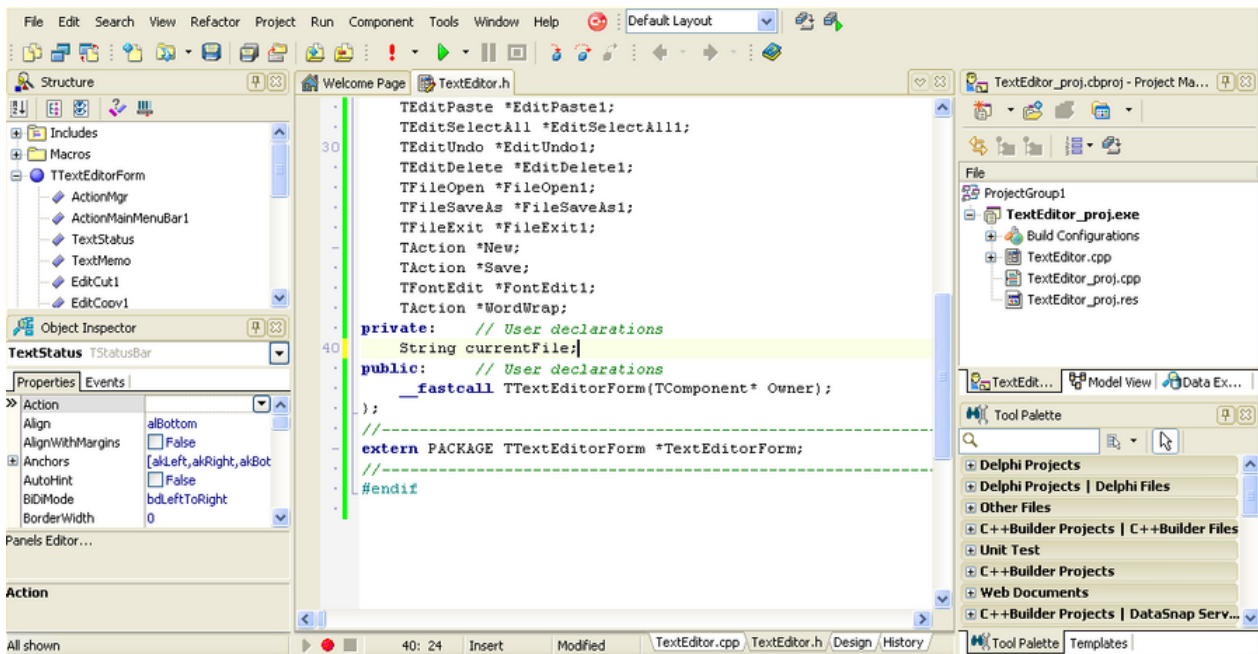


Figure 3-19. Defining the `currentFile` private variable (C++Builder view)

## Creating an event handler for the New command

You are now ready to define the responses to clicking menu items. In the **Form Designer**, click **File > New** on the menu bar in your text editor form. Then select the **Events** tab in the **Object Inspector**, as in the following image. Click the plus sign (+) to expand the **Action** list if necessary.

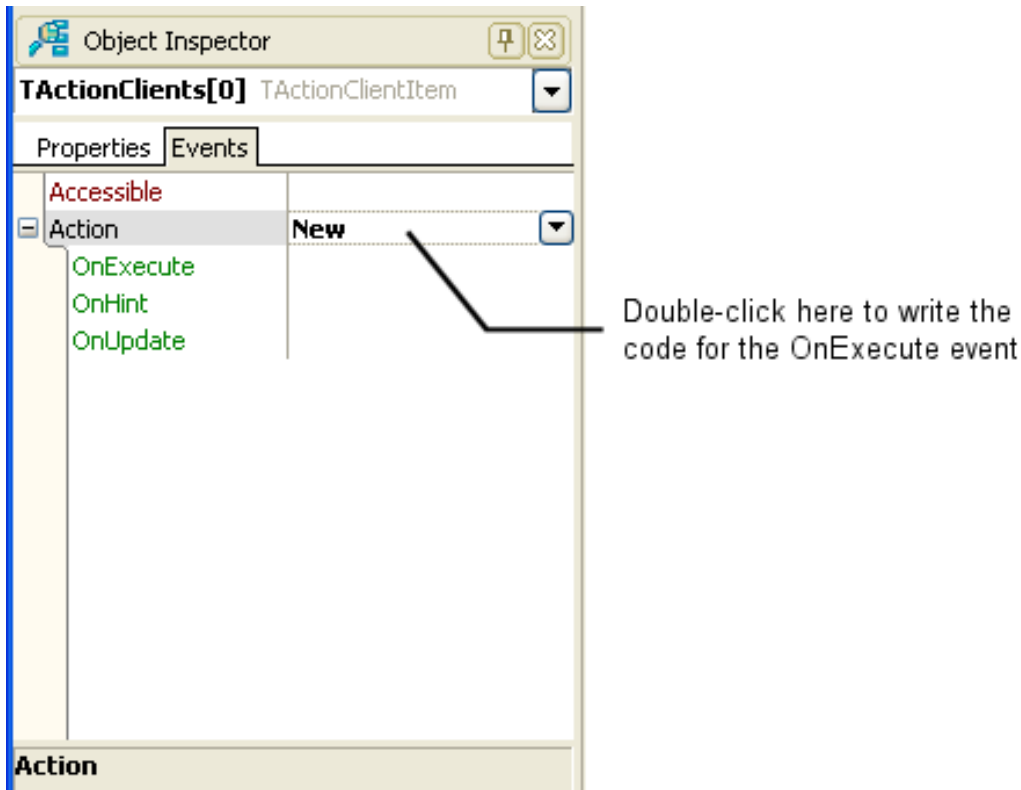


Figure 3-20. Opening the Events tab in the Object Inspector

Double-click the edit box corresponding to the OnExecute event. The **Code Editor** opens and displays the following function skeleton, using Delphi or C++Builder, respectively.

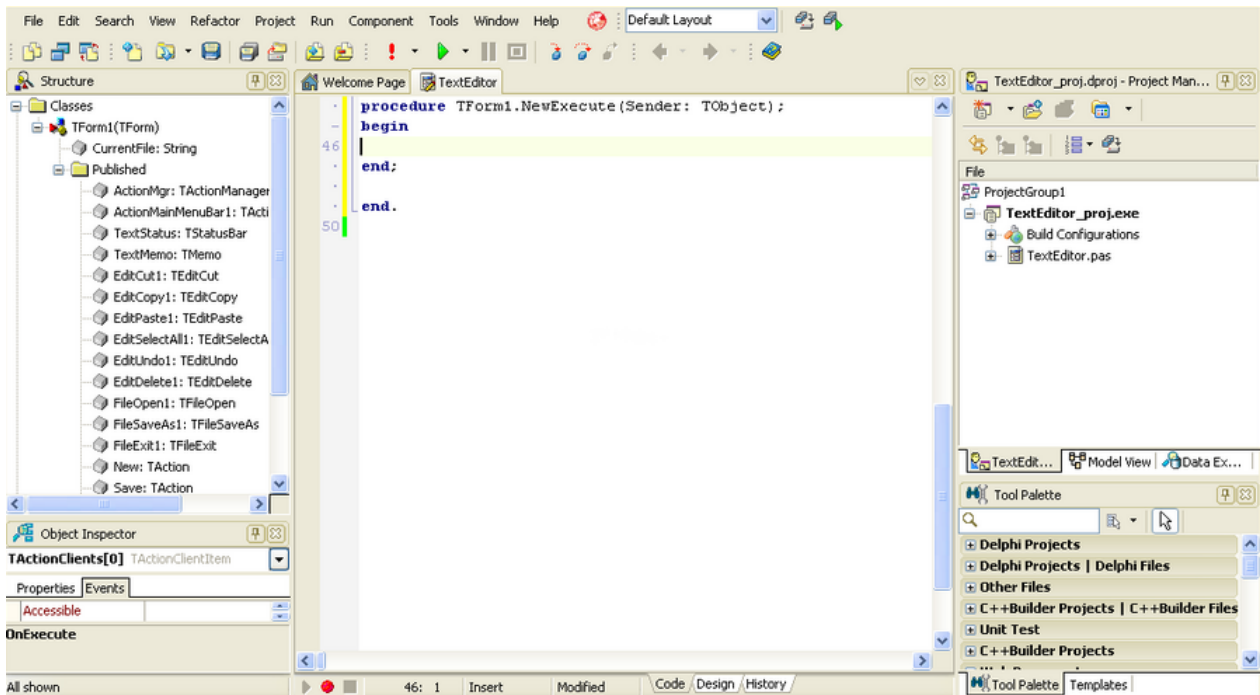


Figure 3-21. Automatic generation of the code skeleton for the OnExecute event (Delphi view)

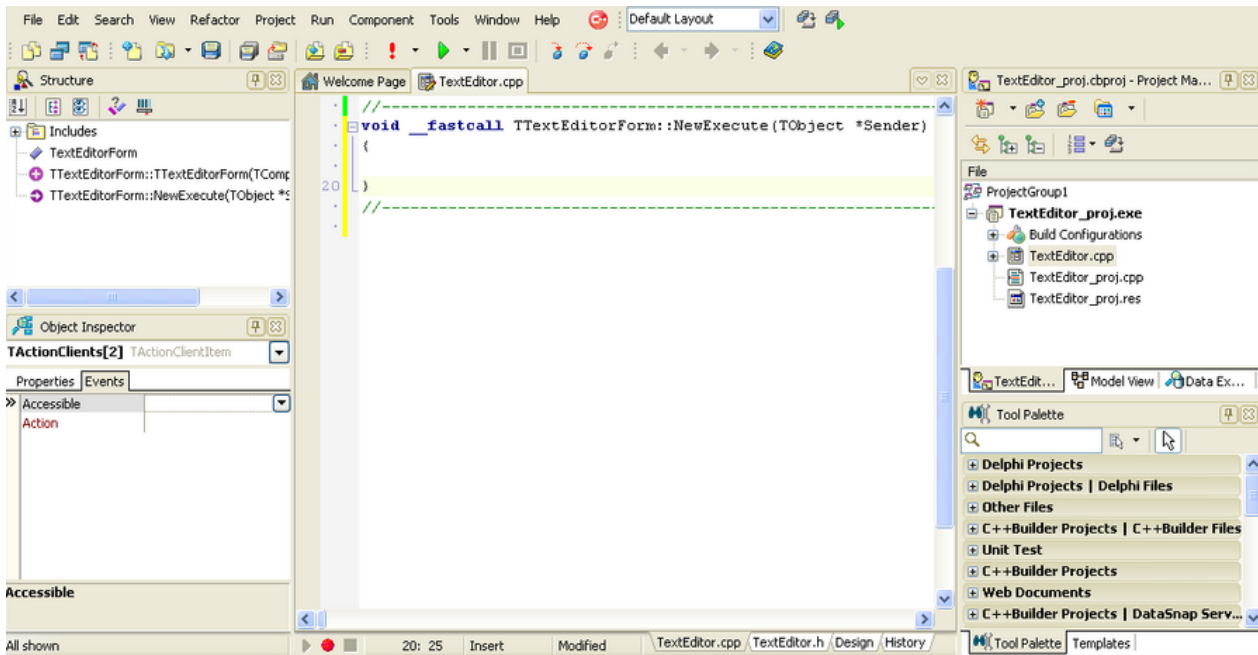


Figure 3-22. Automatic generation of the code skeleton for the OnExecute event (C++Builder view)

Now write the code that executes when the user selects **File > New** inside the code skeleton previously generated, as in the following:

```
// New menu item event handler
procedure TTextEditorForm.NewExecute(Sender: TObject);
var
    UserResponse : Integer;
begin
    // Check TMem number of lines property
    if TextMemo.Lines.Count > 0 then
    begin
        UserResponse := MessageDlg(
            'This will clear the current document. ' + 'Do you want to
continue?' ,
            mtInformation,
            mbYesNo, 0);

        if UserResponse = mrYes then
        begin
            TextMemo.Clear;
            CurrentFile := '';
        end;
    end;
end;

// New menu item event handler
void __fastcall TTextEditorForm::NewExecute(TObject *Sender)
{
    // Check TMem number of lines property
    if (TextMemo->Lines->Count > 0)
    {
```

```

int userResponse = MessageDlg(
String("This will clear the current document. ")
+ "Do you want to continue?", mtInformation,
TMsgDlgButtons() << mbYes << mbNo, 0);

if (userResponse == mrYes) {
    TextMemo->Clear();
    currentFile = "";
}
}
}

```

## Creating the event handlers for the Open command

Return to the form on the **Design** tab. Perform a set of steps similar to creating the handler for the **New** menu item. Click **File > Open** on the menu bar in your text editor form. Select the **Events** tab in the **Object Inspector** and click the plus sign (+) to expand the **Action** list if necessary. Double-click the **OnAccept** event and write the code displayed below.

```

procedure TTextEditorForm.FileOpen1Accept(Sender: TObject);
var
    FileName: String;
begin
    // Get file name from TFileOpen component
    FileName := FileOpen1.Dialog.FileName;
    if FileExists(FileName) then
    begin
        TextMemo.Lines.LoadFromFile(FileName);
        CurrentFile := FileName;
        Self.Caption := 'Text Editor - ' + ExtractFileName(FileName);
    end;
end;

```

```

void __fastcall TTextEditorForm::FileOpen1Accept(TObject *Sender)
{
    // Get file name from TFileOpen component
    String fileName = FileOpen1->Dialog->FileName;
    if (FileExists(fileName)) {
        TextMemo->Lines->LoadFromFile(fileName);
        currentFile = fileName;
        this->Caption = "Text Editor - " + ExtractFileName(fileName);
    }
}

```



## Creating the event handlers for the SaveAs command

Return to the form and double-click the OnAccept event of **File > SaveAs** and write the following code.

```

procedure TTextEditorForm.FileSaveAs1Accept(Sender: TObject);
var
    FileName: String;
    UserResponse : Integer;
begin
    // Get file name from TFileSaveAs component
    FileName := FileSaveAs1.Dialog.FileName;

    if FileExists(FileName) then
    begin
        UserResponse := MessageDlg(
            'File already exists. ' +
            'Do you want to overwrite?' , mtInformation,
            mbYesNo, 0);
        if UserResponse = mrNo then
            Exit();
    end;

    TextMemo.Lines.SaveToFile(FileName);
    CurrentFile := FileName;
    Self.Caption := ExtractFileName(FileName);
end;

void __fastcall TTextEditorForm::FileSaveAs1Accept(TObject *Sender)
{
    // Get file name from TFileSaveAs component
    String fileName = FileSaveAs1->Dialog->FileName;

    if (FileExists(fileName)) {
        int userResponse = MessageDlg(
            String( "File already exists. " ) +
            "Do you want to overwrite?" , mtInformation,
            TMsgDlgButtons() << mbYes << mbNo, 0);
        if (userResponse == mrNo) {
            return;
        }
    }

    TextMemo->Lines->SaveToFile(fileName);
    currentFile = fileName;
    this->Caption = ExtractFileName(fileName);
}

```

## Creating the event handlers for the Save command

Double-click the OnExecute event of **File > Save** and write the following lines of code.

```
procedure TTextEditorForm.SaveExecute(Sender: TObject);
begin
  if CurrentFile = '' then
    Self.FileSaveAs1.Execute()
  else
    TextMemo.Lines.SaveToFile(CurrentFile);
end;

void __fastcall TTextEditorForm::SaveExecute(TObject *Sender)
{
  if (currentFile == "") {
    this->FileSaveAs1->Execute();
  }
  else {
    TextMemo->Lines->SaveToFile(currentFile);
  }
}
```

## Creating the event handlers for the Font command

Double-click the OnAccept event of **Format > Font** and write the following code.

```
procedure TTextEditorForm.FontEdit1Accept(Sender: TObject);
begin
  // Set TMemo font property
  TextMemo.Font := FontEdit1.Dialog.Font;
end;

void __fastcall TTextEditorForm::FontEdit1Accept(TObject *Sender)
{
  // Set TMemo font property
  TextMemo->Font = FontEdit1->Dialog->Font;
}
```

## Creating the event handlers for the Word Wrap command

Next, double-click the OnExecute event of **Format > Word Wrap** and write the following code.

```
procedure TTextEditorForm.WordWrapExecute(Sender: TObject);
begin
  { Toggle the word wrapping state. }
  TextMemo.WordWrap := not TextMemo.WordWrap;
  WordWrap.Checked := TextMemo.WordWrap;
  if TextMemo.WordWrap = True then
    { Only vertical scrollbars are needed when word wrapping is set. }
    TextMemo.ScrollBars := ssVertical
  else
```

```

        TextMemo.ScrollBars := ssBoth;
    end;

void __fastcall TTextEditorForm::WordWrapExecute(TObject *Sender)
{
    // Toggle the word wrapping state.
    TextMemo->WordWrap = !TextMemo->WordWrap;
    WordWrap->Checked = TextMemo->WordWrap;
    if (TextMemo->WordWrap == True) {
        // Only vertical scrollbars are needed when word wrapping is set.
        TextMemo->ScrollBars = ssVertical;
    }
    else {
        TextMemo->ScrollBars = ssBoth;
    }
}

```

## Creating event handlers for the status bar

Use the status bar to display the current cursor position and the number of lines in the open text file. Double-click the `OnMouseDown` event of the `TextMemo` component and write the following code, in Delphi and C++ respectively. The `CaretPos` property is used to indicate the coordinates of the cursor inside the text memo box.

```

procedure TTextEditorForm.TextMemoMouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    textStatus.Panels.Items[ 0 ].Text :=
        'L: ' + IntToStr(TextMemo.CaretPos.Y + 1);
    textStatus.Panels.Items[ 1 ].Text :=
        'C: ' + IntToStr(TextMemo.CaretPos.X + 1);
    textStatus.Panels.Items[ 2 ].Text :=
        'Lines: ' + IntToStr(TextMemo.Lines.Count);
end;

void __fastcall TTextEditorForm::TextMemoMouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y)
{
    textStatus->Panels->Items[ 0 ]->Text =
        "L: " + String (TextMemo->CaretPos.y + 1);
    textStatus->Panels->Items[ 1 ]->Text =
        "C: " + String (TextMemo->CaretPos.x + 1);
    textStatus->Panels->Items[ 2 ]->Text =
        "Lines: " + IntToStr (TextMemo->Lines->Count);
}

```

Finally, double-click the `OnKeyDown` event of `TextMemo` and write the code below. The `OnKeyDown` event is triggered whenever you press a key inside the text memo box.

```

procedure TTextEditorForm.TextMemoKeyDown(Sender: TObject;
    var Key: Word; Shift: TShiftState);

```

```

begin
    // Perform same action as for mouse down in TMemo
    TextMemoMouseDown(Sender, mbLeft, Shift, 0, 0);
end;

void __fastcall TTextEditorForm::TextMemoKeyDown(TObject *Sender,
WORD &Key, TShiftState Shift)
{
    // Perform same action as for mouse down in TMemo
    TextMemoMouseDown(Sender, mbLeft, Shift, 0, 0);
}

```

## Next

Compiling and running the application

# Compiling and running the application (IDE Tutorial)

*Go Up to Starting your first RAD Studio application Index (IDE Tutorial)*

Before you can actually see your application running, you must first compile and build it. To compile your application, press CTRL-F9 or select **Project > Compile TextEditor\_proj**. To build your application, press SHIFT-F9 or select **Project > Build TextEditor\_proj**. You then see a dialog box displaying the progress of the compilation or build. If your application contains any syntactical errors, you will have to correct them and then recompile it.

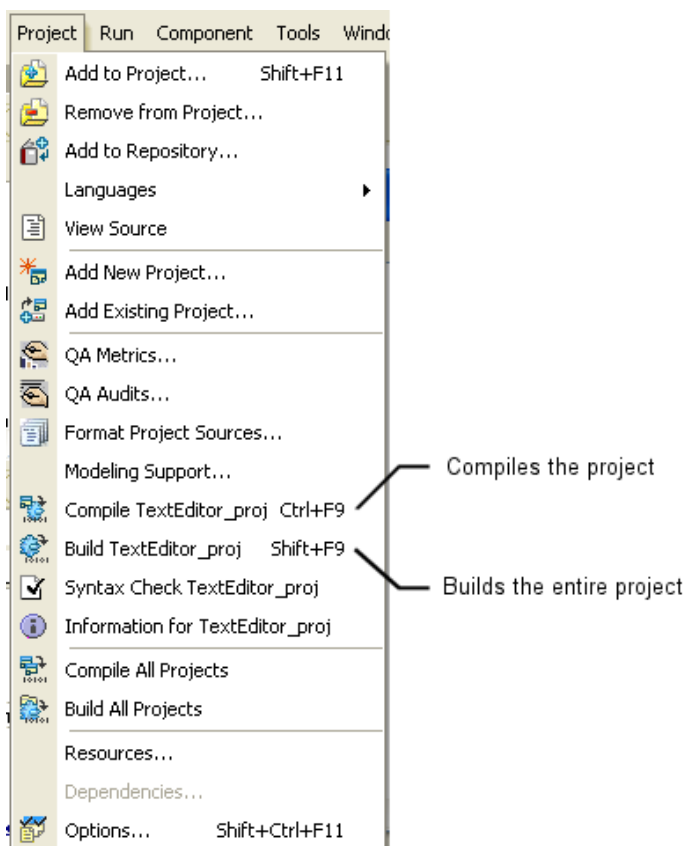


Figure 3-23. Project menu options for compiling and building the project

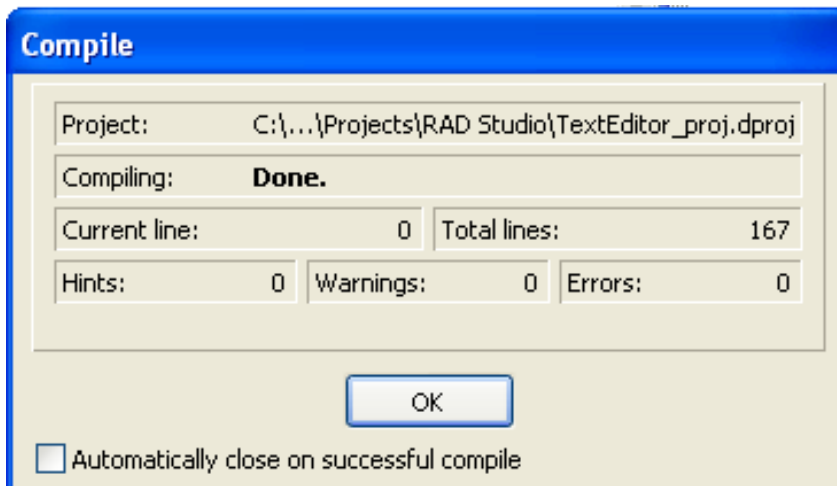


Figure 3-24. Dialog showing the success of compiling the application (Delphi)

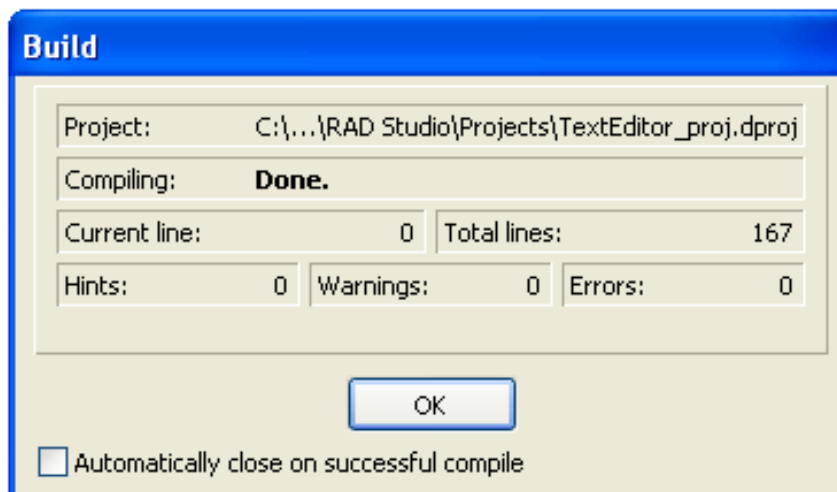


Figure 3-25. Dialog showing the success of building the application (Delphi)

After you build the application, you can see how it behaves at run time. Press F9 or click **Run > Run** to run your application in debug mode.

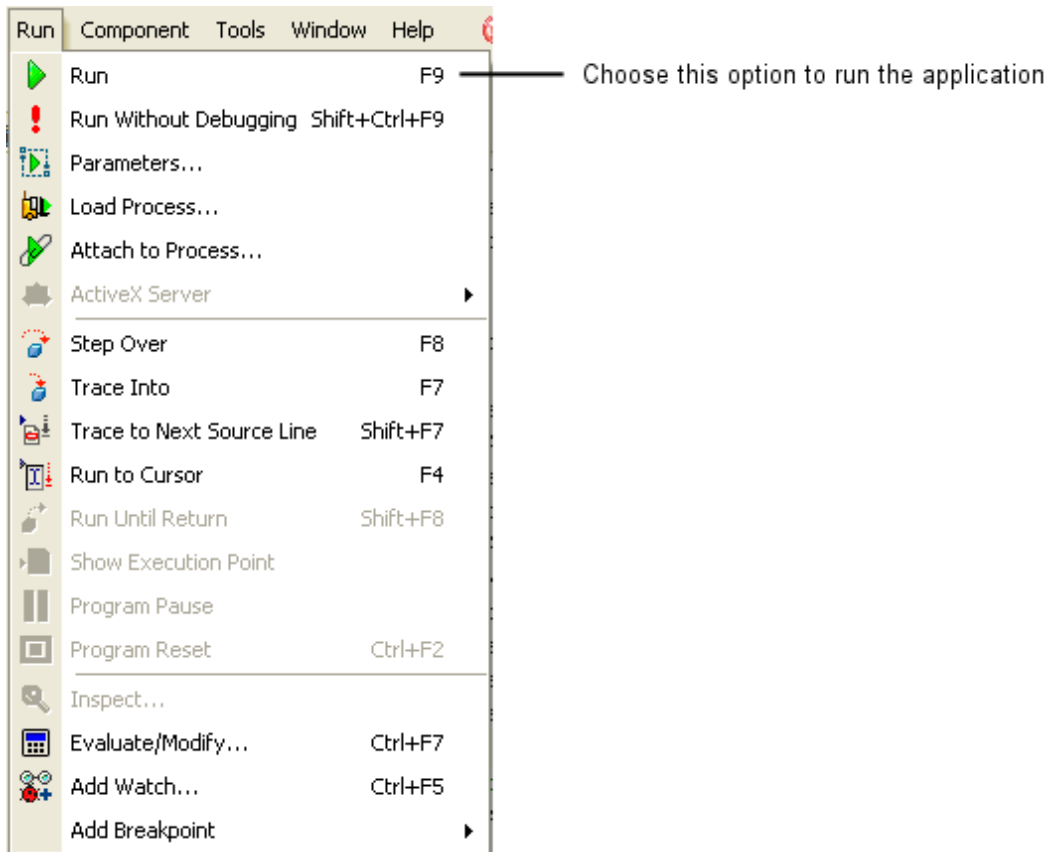


Figure 3-26. Running the application from the Run menu

There are a few other options available, but these are beyond the scope of this tutorial.

**Note:** You can directly run the application without compiling it first. RAD Studio automatically detects whether compilation is required and compiles the project if necessary.


Even if the application successfully compiles and runs, it might still not perform as you intended. The next section, Debugging the application, describes how to use some of the RAD Studio debugging features to rapidly find and fix bugs.

## Next

Debugging the application

# Debugging the application (IDE Tutorial)

Go Up to [Starting your first RAD Studio application Index \(IDE Tutorial\)](#)

To get a glimpse of the basic debugging features in RAD Studio, set a breakpoint on the first line of the `FileSaveAs1Accept` function by clicking on the bar at the left of the line of code. You can place a breakpoint only on executable lines of code, marked by blue circles , as seen in the figures below.

The **Code Editor** window should look like the following figure.

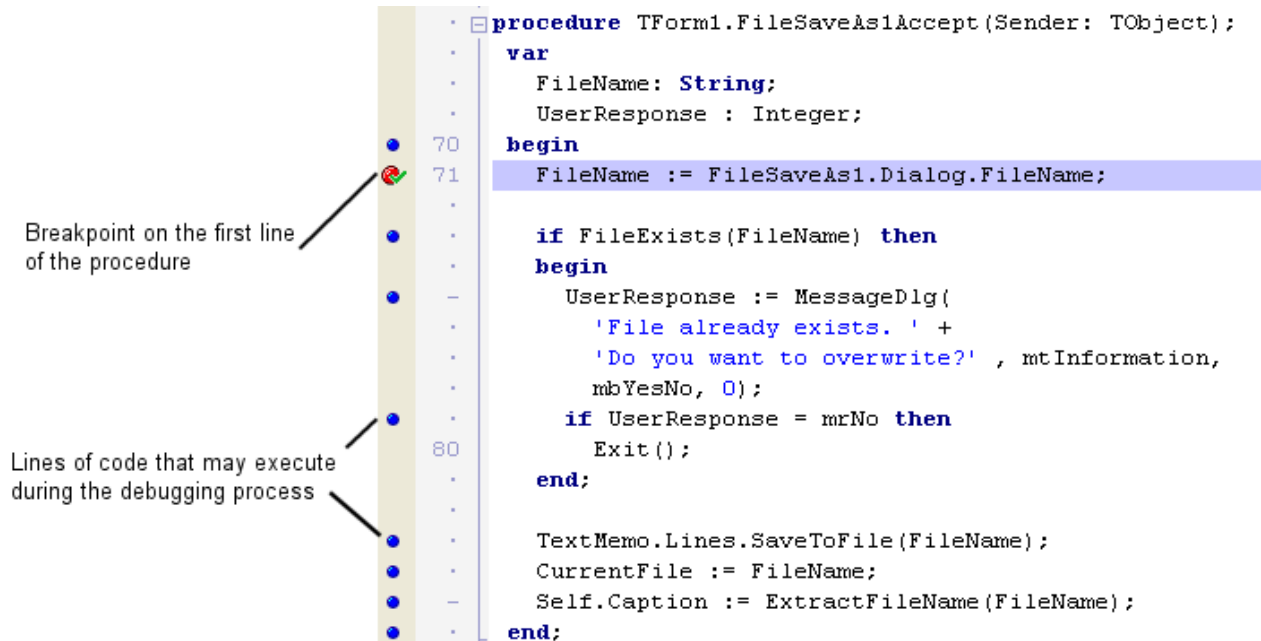


Figure 3-27. Debugging the `FileSaveAs1Accept` procedure (Delphi code)

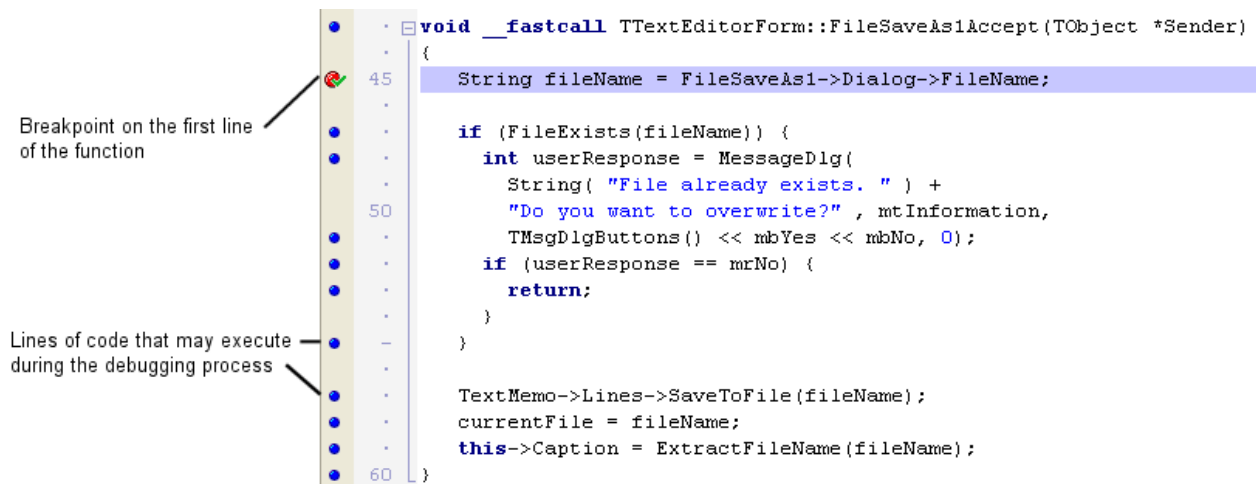


Figure 3-28. Debugging the `FileSaveAs1Accept` function (C++Builder code)

Press F9 to run the application, type something in the text box of the text editor and click **File > Save As**. Name the new text file, making sure that a file with the same name does not already exist in that folder. After clicking **Save**, the application should stop at the breakpoint you have previously set, and the code editor window should display as in the screenshot below.

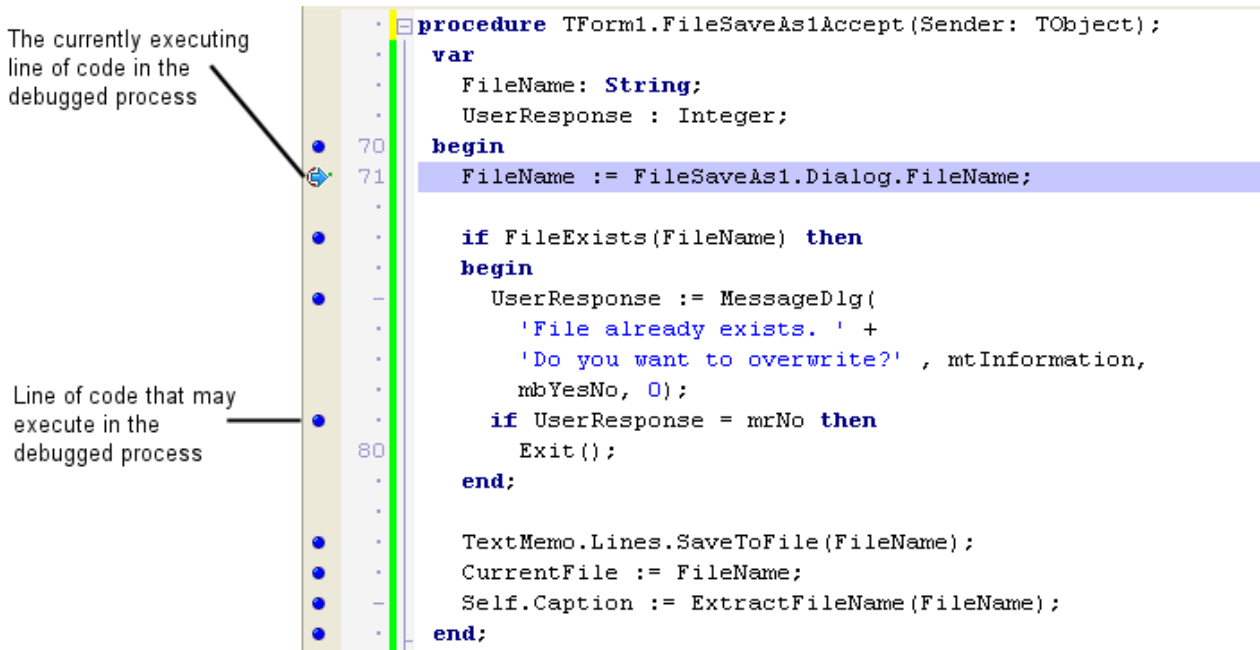


Figure 3-29. Application stopping at the specified breakpoint (Delphi view)

To see the value of the FileName variable, select the FileName word in the first line of FileSaveAs1Accept and drag it to the **Watch List**, as in the following figures.

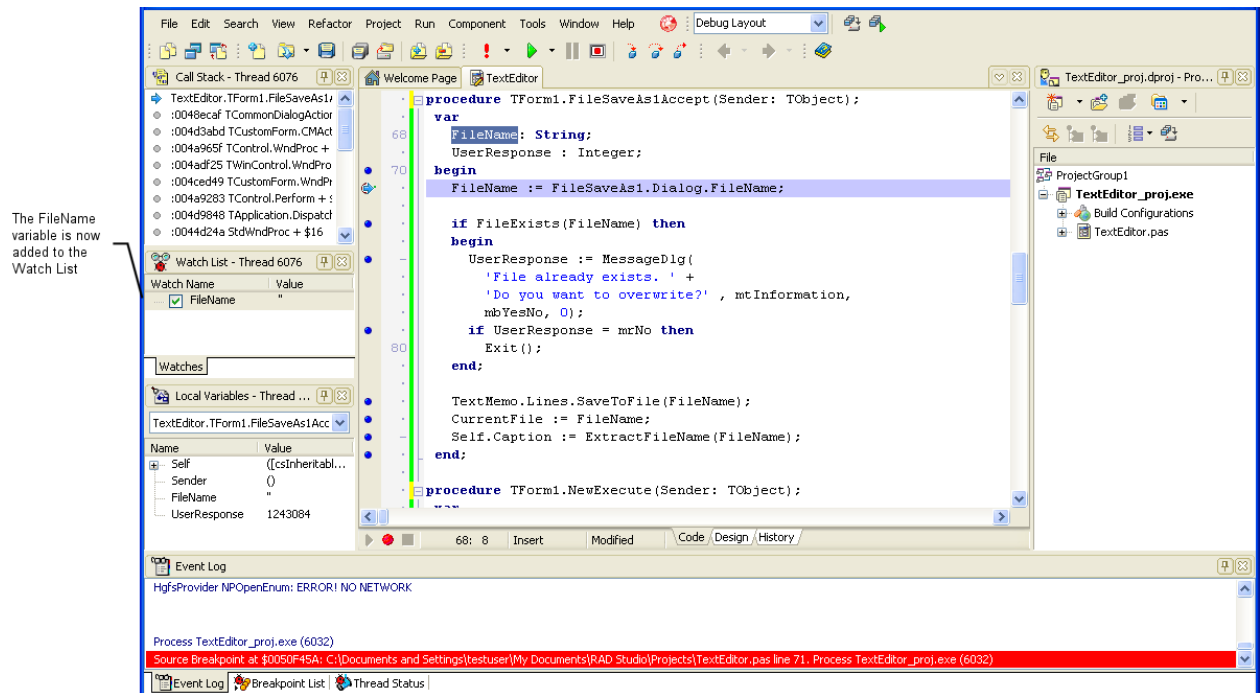


Figure 3-30. Dragging the FileName variable to the Watch List (Delphi view)



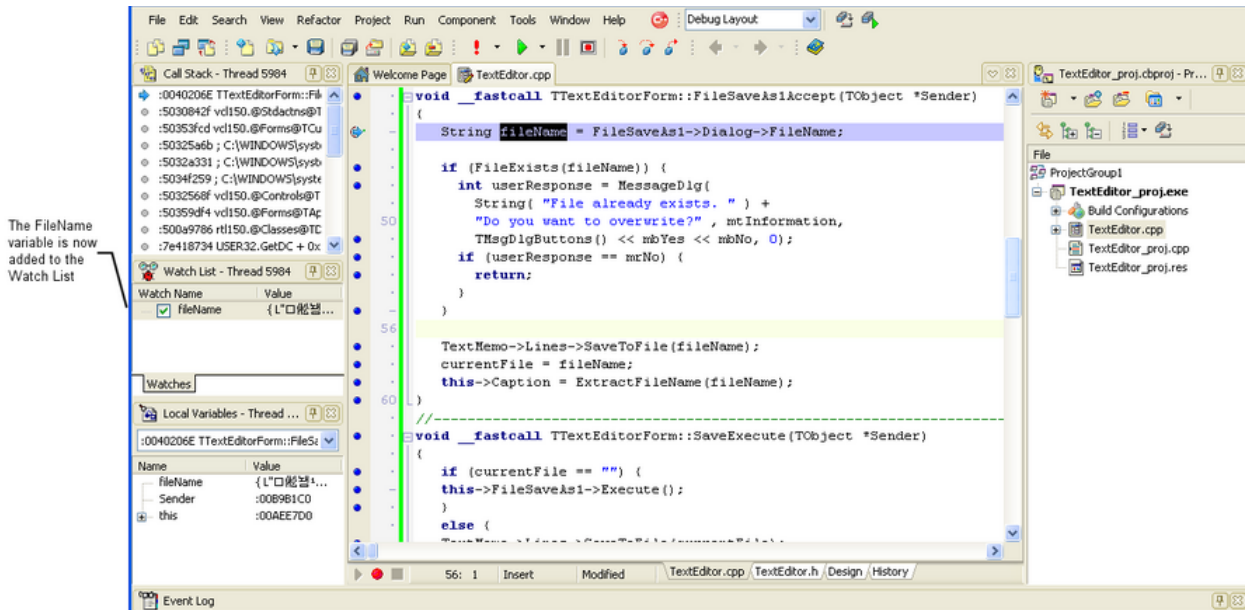


Figure 3-31. Dragging the `fileName` variable to the Watch List (C++ Builder view)

Press F8 to advance to the following line of code, so that the value of `FileName` is updated. To expand the value of `FileName`, hover the mouse cursor over its label in the **Watch List** and wait. The result should look as in the following figures.

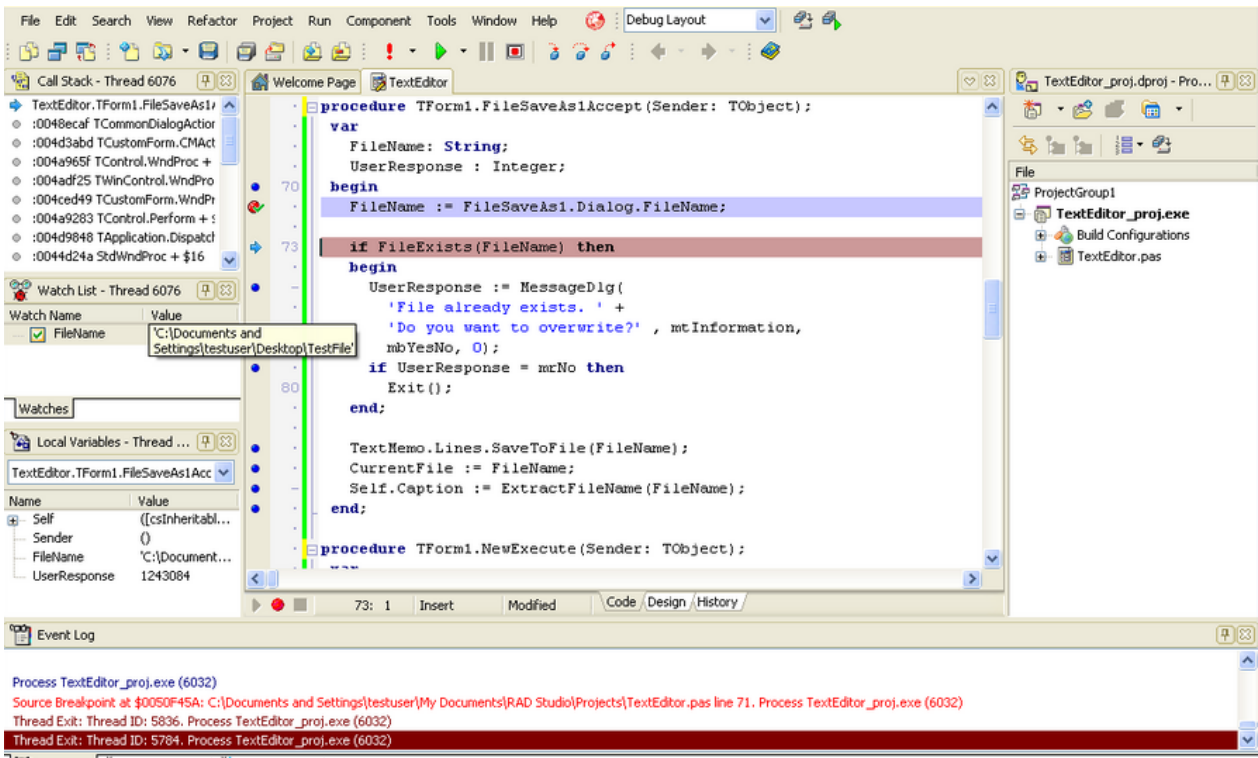


Figure 3-32. Advancing to the next line of code to change the value of `FileName` (Delphi view)

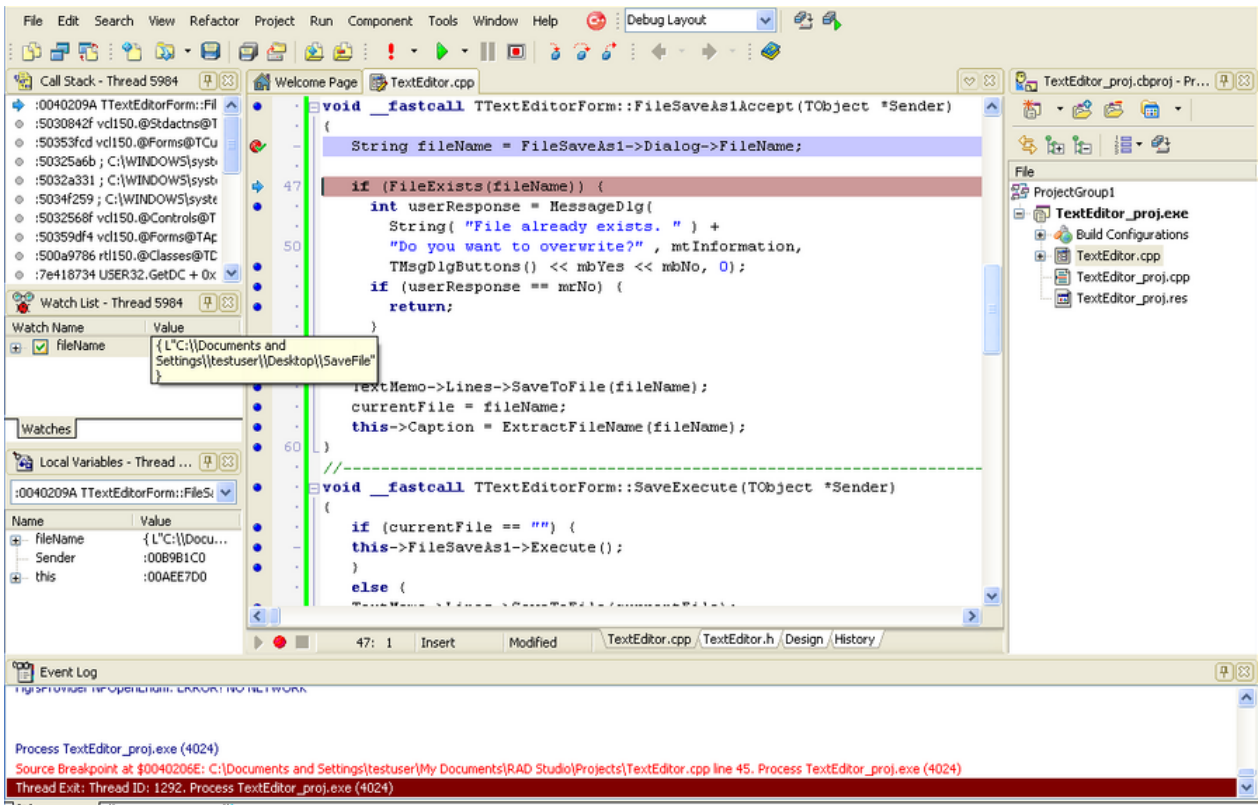


Figure 3-33. Advancing to the next line of code to change the value of FileName (C++ Builder view)

Pressing F8 one more time jumps over the if statement, since a file with the given name does not already exist at the current location. The screen should now look like the following images.

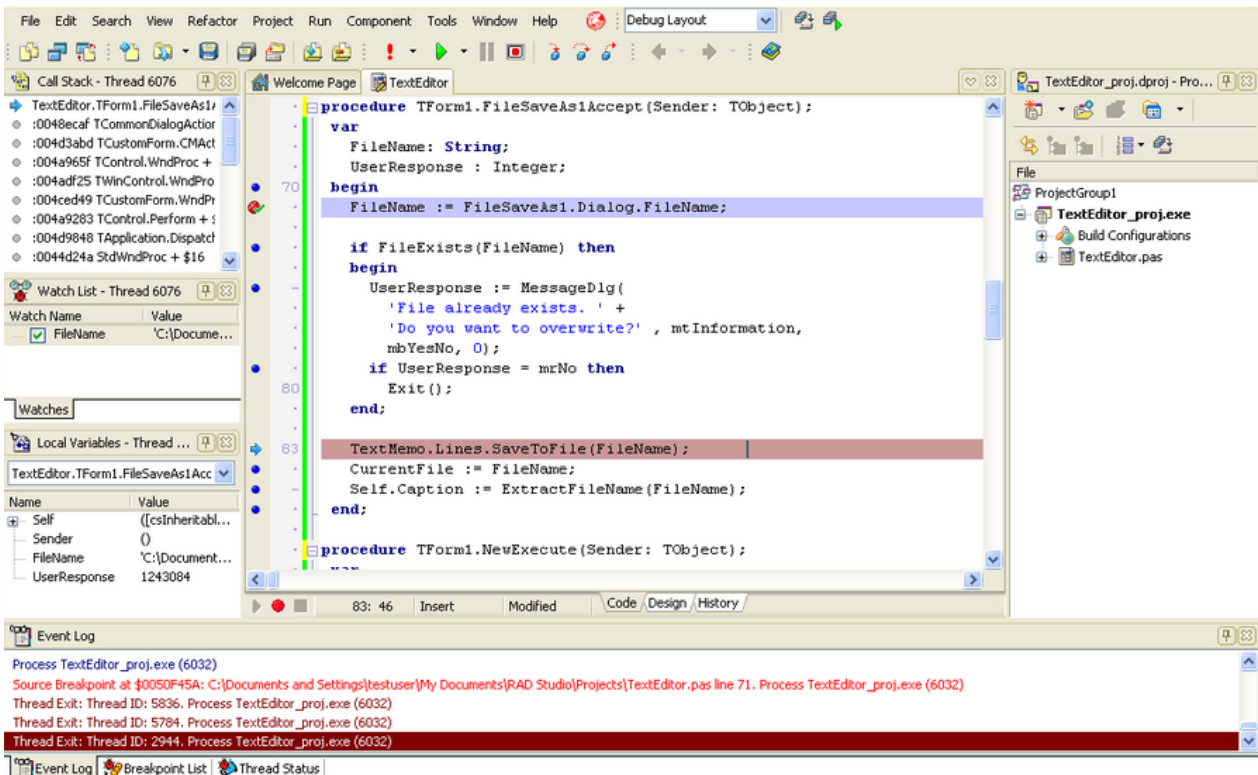


Figure 3-34. Jumping over the if statement (Delphi view)

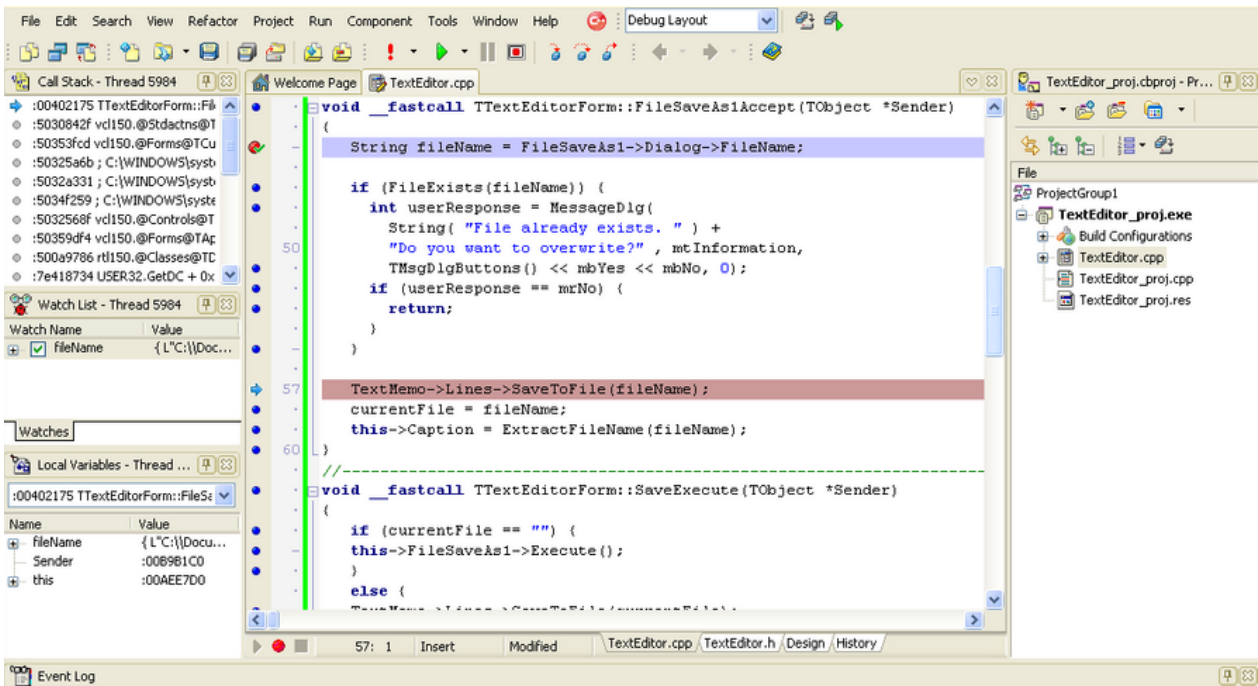


Figure 3-35. Jumping over the if statement (C++Builder view)

Press F8 until you get to the last line of the FileSaveAs1Accept function. Now move the mouse cursor over the name of the CurrentFile variable to instantly view its value, as in the following images.

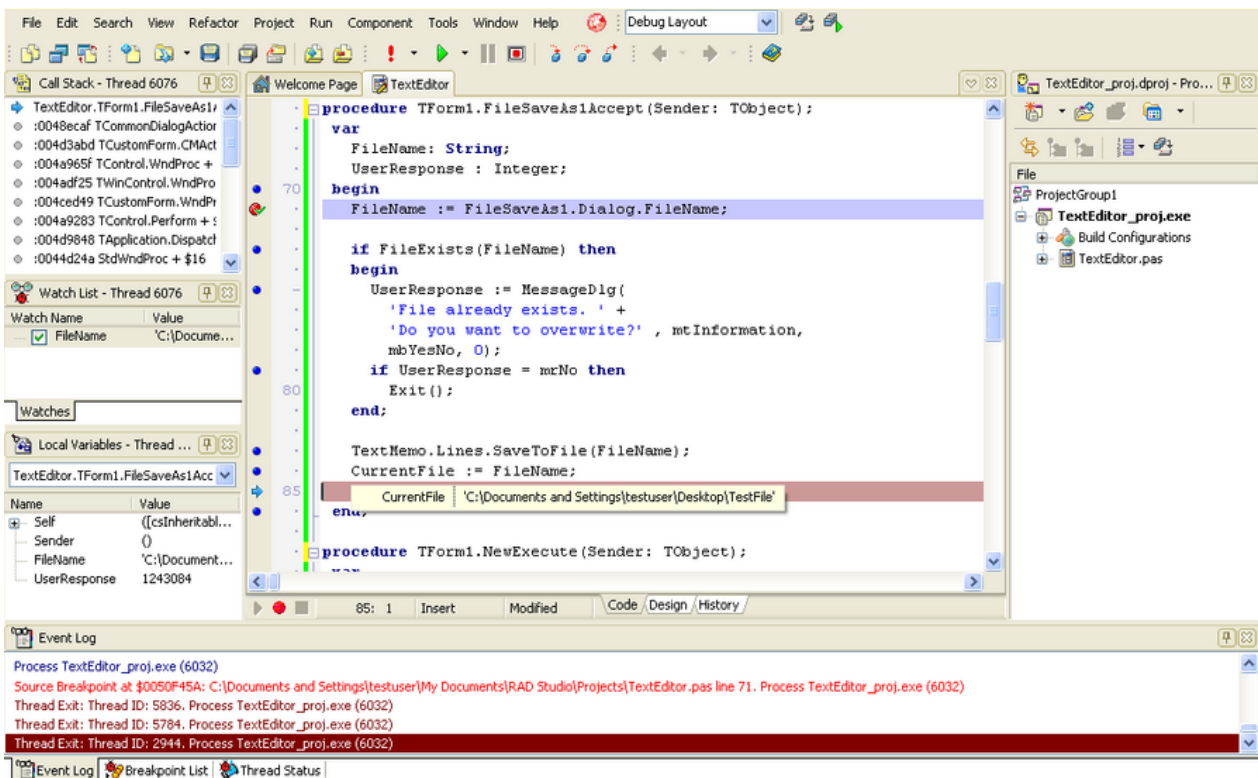


Figure 3-36. Viewing the value of CurrentFile (Delphi view)

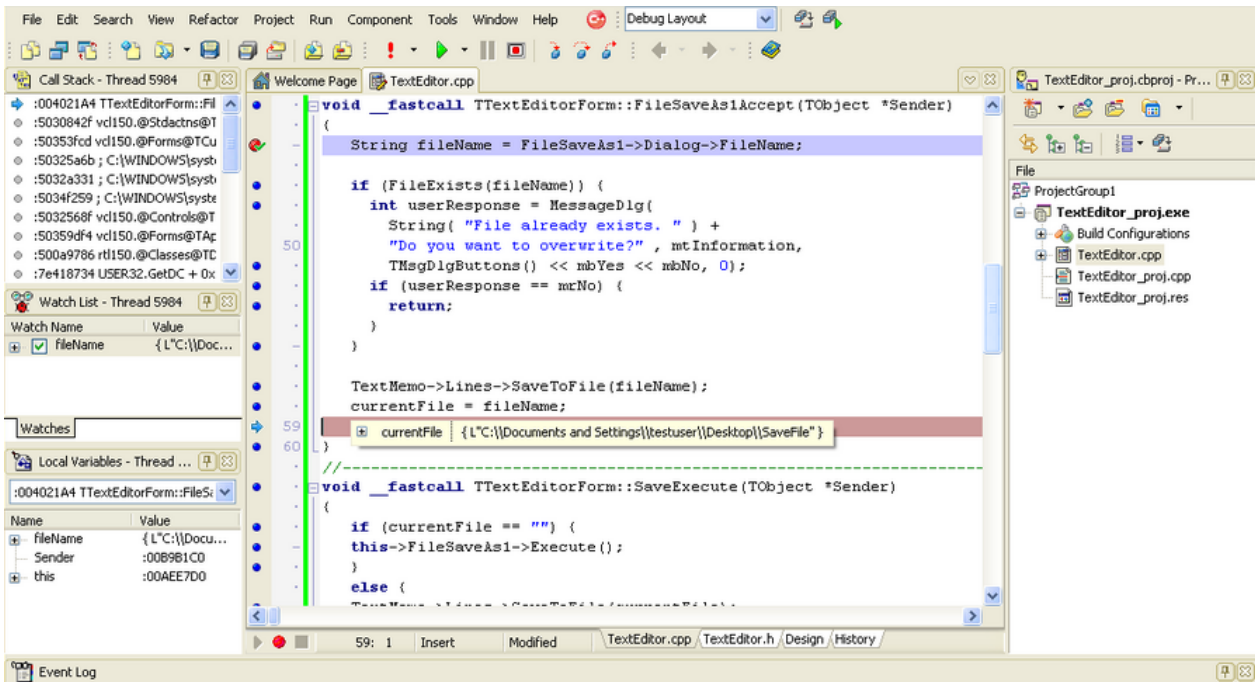


Figure 3-37. Viewing the value of CurrentFile (C++Builder view)

To end the debugging session, press the stop button on the Debug toolbar, also displayed in the following figure.



Click here to stop debugging the application

Figure 3-38. The Debug toolbar

## Next

More advanced topics

# More advanced topics Index (IDE Tutorial)

---

*Go Up to Tutorial: Using the IDE for Delphi and C++Builder*

## Topics

- VCL and RTL
- Third party add-ins

## VCL and RTL (IDE Tutorial)

---

*Go Up to More advanced topics Index (IDE Tutorial)*

As seen in previous sections, Embarcadero RAD Studio offers a powerful Integrated Development Environment that makes building native Windows applications extremely easy. The Visual Component Library (also known as VCL) offers a large number of visual and nonvisual components that can be used to build almost any desired user interface. Besides the VCL, RAD Studio provides an extensive library of routines and classes, called the Run Time Library (known as RTL), that provides the common functionality needed in all applications.

This topic lists the most important classes, data types, and functions that can be found both in the VCL and RTL.

The most important components of the VCL are:

- A standard set of components that include all controls provided by the Windows UI framework. This set consists of components such as buttons, edits, menus, and so on. The VCL also extends some of these controls, offering you even more functionality than is normally provided by the Windows controls.
- An extended set of components not normally present in the Windows UI framework. These components are built on top of the standard set.
- *Actions*, which is a key concept extensively used in VCL applications, allow you to centralize all the interaction logic of your user interface.
- A number of data-aware controls that can be linked to a data source at design time. These components are widely used in database applications.
- Ribbon Controls that allow you to build the next generation of user interfaces that integrate nicely with the Windows Vista, Windows 7 and Microsoft Office 2007 look-and-feel.
- dbExpress and dbGo database frameworks. These frameworks can be used with all the data-aware controls, simplifying your application development more than ever.
- *Internet Direct*, also known as *Indy*, that provides an extensive number of components used in Internet-connected applications. Indy includes client and server components for today's most used communication protocols on the Internet.
- DataSnap, which allows you to build distributed applications.
- Easy integration of any exposed OLE and ActiveX objects in your application. RAD Studio provides a set of tools that allow creating a wrapper component over any public ActiveX. This wrapper component can be used as any normal VCL component inside your application.

Even though this is not the full list of components available in the VCL, the above mentioned are the most widely used and appreciated VCL components. To see all available components, check out the **Tool Palette** in RAD Studio.

For more information on VCL, see [VCL Overview](#).

The most useful features in the RTL, which are available both in Delphi and C++Builder, are:

- Extensive support for strings. This support includes handling of Unicode strings (the default encoding used by RAD Studio), ANSI and UTF-8 strings, various string handling routines, and much more.

- A large number of date and time manipulation routines.
- Extensive support for file and stream operations.
- Routines and classes that provide Windows API support. You, as a developer, are often required to use Windows API directly because a certain functionality is not provided by the RTL. RAD Studio provides developers with the ability to use the full Windows API directly. However, RAD Studio also provides classes, such as **TRegistry** for registry handling, that provide Windows API capabilities but are easier to use.
- Variant data types and various support routines to make COM integration easy. Variant data types have long been used in Microsoft COM and OLE technologies. Variants are useful when you do not know the exact data type you are operating on. The Delphi language compiler provides native support for Variants, integrating some of the dynamic language concepts, found in other languages such as Java, PHP, and others.
- Run-time Type Information, also known as RTTI, that provides an easy way to obtain metadata about types, classes, and interfaces at runtime.

Another important part of the RTL is provided by the generic collections, which is specific to the Delphi language. This collection of generic classes can be used in any application that requires lists, dictionaries, and other container classes. There are also nongeneric counterparts for these classes.

The C++Builder equivalent of the generic collections is given by the STL library, provided by Dinkumware as a third party add-in. The next section elaborates on this.

*For more information...*

See Win32 Developer's Guide.

## Next

Third party add-ins

---

# Third party add-ins (IDE Tutorial)

---

*Go Up to More advanced topics Index (IDE Tutorial)*

RAD Studio includes software provided by third parties.

## AQTime

*SmartBear's AutomatedQA AQtime* is an award-winning performance profiling and memory and resource debugging toolset for Microsoft, Embarcadero, Intel, Compaq and GNU compilers.

Get more information about **SmartBear Software**, at <http://www.automatedqa.com/products/aqtime/>.

## Beyond Compare

*Beyond Compare* allows you to quickly and easily compare your files and folders. You can merge changes, synchronize files, and generate reports for your records.

See **Scooter Software** at <http://www.scootersoftware.com>.

## Boost

*Boost* is a set of libraries for C++ that significantly expand the language using template meta-programming. A fully tested and preconfigured subset of Boost is included in C++Builder. Include paths have been set for the Boost libraries, and any other necessary libraries should be automatically linked. To use the Boost minmax library, for instance, your code should include:

```
#include <boost/algorithm/minmax.hpp>
```

See the **Boost Open Source Project** at <http://www.boost.org>.

## CodeSite

The *CodeSite Logging System* gives developers deeper insight into how their code is executing, which enables them to locate problems more quickly and ensure their application is running correctly.

For more information on CodeSite by **Raize Software**, see <http://www.raize.com/devtools/codesite/>.

## Dinkumware (STL)

*Dinkumware (STL)* is a collection of template libraries for C++ included in C++Builder. The libraries include containers such as vectors, lists, sets, maps, bitsets. Dinkumware also includes algorithms for common operations such as sorting a container or searching inside a container. To implement the algorithms, STL includes iterators in five flavors for operating on a container: input, output, forward, backward and bidirectional. Functors, or function objects, are also available for overloading operators.

See **Dinkumware Ltd**, at <http://www.dinkumware.com>.

## FinalBuilder

*FinalBuilder* is a powerful Automated Build & Release Management tool that simplifies software build automation.

See <http://www.finalbuilder.com> for more information.

## InstallAware

*InstallAware* focuses on state of the art software installation tools that offer the highest compression ratios and bullet-proof installations. The company produces software installation and compression technologies for the Windows Installer (MSI) platform on Microsoft Windows.

See **InstallAware** at <http://www.installaware.com>.

## Internet Direct (Indy)

*Internet Direct (Indy)* is an open source group. The Indy project maintains several active open source projects that have evolved from the original Indy project. Indy offers client and server components using Internet protocols, such as TCP, UDP, echo, FTP, HTTP, Telnet and many others. Indy also provides components for I/O handling, intercepts, SASL, UUE, MIME, XXE encoders and others.

See the **Indy Project** at <http://www.indyproject.org>.

## Intraweb

*Intraweb* is a collection of visual components, a framework designed to allow you to create web applications or Apache plug-ins. Intraweb allows you to create web applications with the same ease you use the VCL.

See **ATOZED Software** for IntraWeb at <http://www.atozed.com/IntraWeb>.

## IP\*Works!

*IP\*Works!* eliminates the complexity of Internet development, providing easy-to-use, programmable components that facilitate tasks such as sending email, transferring files, managing networks, browsing the web and consuming web services.

See **/n software**, at <http://www.nsoftware.com/ipworks/>

## Rave Reports

**Rave Reports** by Nevrona Designs offers a wide range of reporting solutions, such as rock solid reporting for corporate environments through a highly scalable reporting server, powerful application reporting tools for software developers, and flexible and easy to use information reporting applications for end users.

See **Nevrona** at <http://www.nevrona.com>.

## Subversion Client and Server

*Subversion* by CollabNet is the new standard for version control and Software Configuration Management (SCM) for globally distributed organizations. Also, **Subversion** is integrated into RAD Studio, so you can use Subversion features in the IDE.

See **CollabNet** at <http://www.collab.net>.

## TeeChart

*TeeChart*, by Steema Software, provides complete, quick and easy-to-use charting, .NET, Java, ActiveX / COM, PHP and Delphi VCL controls for business, real-time, financial and scientific applications.

See **Steema Software** at <http://www.steema.com>.

## Next

Other resources



---

## Other resources Index (IDE Tutorial)

---

*Go Up to Tutorial: Using the IDE for Delphi and C++Builder*

A variety of resources are available for developers using RAD Studio. These include support from the Embarcadero Developer Network as well as Embarcadero partners whose products are included in RAD Studio.

### Topics

- Embarcadero Developer Network
- Partners

---

## Embarcadero Developer Network (IDE Tutorial)

---

*Go Up to Other resources Index (IDE Tutorial)*

The Embarcadero Developer Network (EDN), located at [dn.embarcadero.com](http://dn.embarcadero.com) <sup>[1]</sup>, is a collection of code-related articles on various products, including 3rdRail, Turbo Ruby, C++Builder, Delphi, Delphi for PHP, Delphi Prism, Interbase, and JBuilder.

The EDN website keeps an up-to-date calendar of the most important events related to Embarcadero products and also gives the latest news in product updates. As a feature of this website, the calendar can be customized to show the events concerning a single Embarcadero product.

An important developers' resource is the **CodeCentral** page, a part of the EDN, located at the following link: [cc.embarcadero.com](http://cc.embarcadero.com) <sup>[2]</sup>. CodeCentral is a collection of code snippets contributed by various members, using all the programming languages featured in EDN.

### Next

Partners

### References

[1] <http://dn.embarcadero.com>

[2] <http://cc.embarcadero.com>

# Partners (IDE Tutorial)

---

*Go Up to Other resources Index (IDE Tutorial)*

For information on the companies behind all included third party add-ins, see [Third party add-ins](#).

A complete list of the Embarcadero RAD Studio partners can be found at the following links:

- <http://cc.embarcadero.com/partners/delphicpp2009/CBuilder/index.html>
- <http://cc.embarcadero.com/partners/delphicpp2009/Prism/index.html>
- <http://cc.embarcadero.com/partners/delphicpp2009/Delphi/index.html>