

GETTING STARTED WITH WINDOWS AND MAC DEVELOPMENT

The Integrated Development Environment (IDE)

E-Learning Series Course Book – Lesson 3

Embarcadero Technologies

© Copyright 2012 Embarcadero Technologies, Inc. All Rights Reserved.

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

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

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

May 17: Lesson 3 – The Integrated Development Environment

Version: 1.1

Last Updated: May 21, 2012

Presented: May 23, 2012

Prepared by: David Intersimone “David I”, Embarcadero Technologies

© Copyright 2012 Embarcadero Technologies, Inc. All Rights Reserved.

davidi@embarcadero.com

<http://blogs.embarcadero.com/davidi/>

Contents

May 17: Lesson 3 – The Integrated Development Environment	2
Introduction	4
The Main Menu and Toolbars.....	6
The View menu	7
Creating your own custom IDE layouts.....	8
Customizing the File > New menu	9
The Welcome Page and Creating Favorite Project Groups.....	10
Using Project Templates from the Object Repository	13
Customizing the Main Form of your Application using the Object Inspector and Code Editor.....	15
Adding Components to your Application Using the Form Designer and Tool Palette	18
Finding Components using the Tool Palette.....	18
Finding Components using the Tool Palette’s Search Box.....	18
Finding Components using IDE Insight.....	19
Compositing Components.....	20
Customizing Components using the Object Inspector, Property Editors and Component Editors	21
Using the Object Inspector	21
Property Editors	23
Component Editors	24
Code Editor and History Lists	26
The Code Editor.....	26
Change Bars	26
Indenting Code.....	27
Formatting Code	27

E-Learning Series: Getting Started with Windows and Mac Development

Code Insight	28
Code Parameter Hints.....	28
Code Hints.....	29
Help Insight	29
Code Completion	30
Class Completion.....	30
Block Completion	30
Code Browsing (Ctrl-Click).....	31
Code Navigation.....	31
Method Hopping.....	31
Finding Classes	32
Finding Units	32
Finding the Next and Previous Changes	32
Searching Source Code for Usages.....	32
Live Code Templates	32
Code Folding	33
Refactoring.....	34
Sync Edit.....	35
To-Do Lists.....	35
Keystroke Macros	36
Bookmarks	36
Block Comments	36
History List.....	36
The Structure View, Delphi Class Explorer and C++ Class Explorer	38
Structure View	39
Delphi Class Explorer.....	40
C++ Class Explorer	42
Project Manager, Build Configurations and Option Sets	45
Project Manager.....	45
Build Configurations.....	48
Option Sets.....	49
Compiling and Running the Application	51

E-Learning Series: Getting Started with Windows and Mac Development

Debugging the Application.....	52
Stepping Through Code.....	53
Evaluate Modify	53
Breakpoints	54
To set a source breakpoint.....	54
To set an address breakpoint.....	56
To set a data breakpoint	56
To set a module load breakpoint	57
To persist breakpoints from session to session	58
To modify a breakpoint.....	58
To create a breakpoint group	59
To enable or disable a breakpoint or a breakpoint group	59
To create a conditional breakpoint.....	59
To set a breakpoint on a specific thread.....	60
To associate actions with a breakpoint.....	60
To change the color of the text at the execution point or the color of breakpoints.....	60
Watches	60
Debug Windows	61
Summary, Looking Forward, To Do Items, Resources, Q&A and the Quiz	64
To Do Items.....	64
Links to Additional Resources	64
Q&A:.....	64
Self Check Quiz.....	69
Answers to the Self Check Quiz:	70

Introduction

In lesson 2 you built your first Windows and Mac desktop applications using the Integrated Development Environment (IDE), Platform Assistant (PAServer), the Delphi or C++ compiler and FireMonkey. Delphi developers also used the Export to Xcode tool, and Xcode to build their first iOS application.

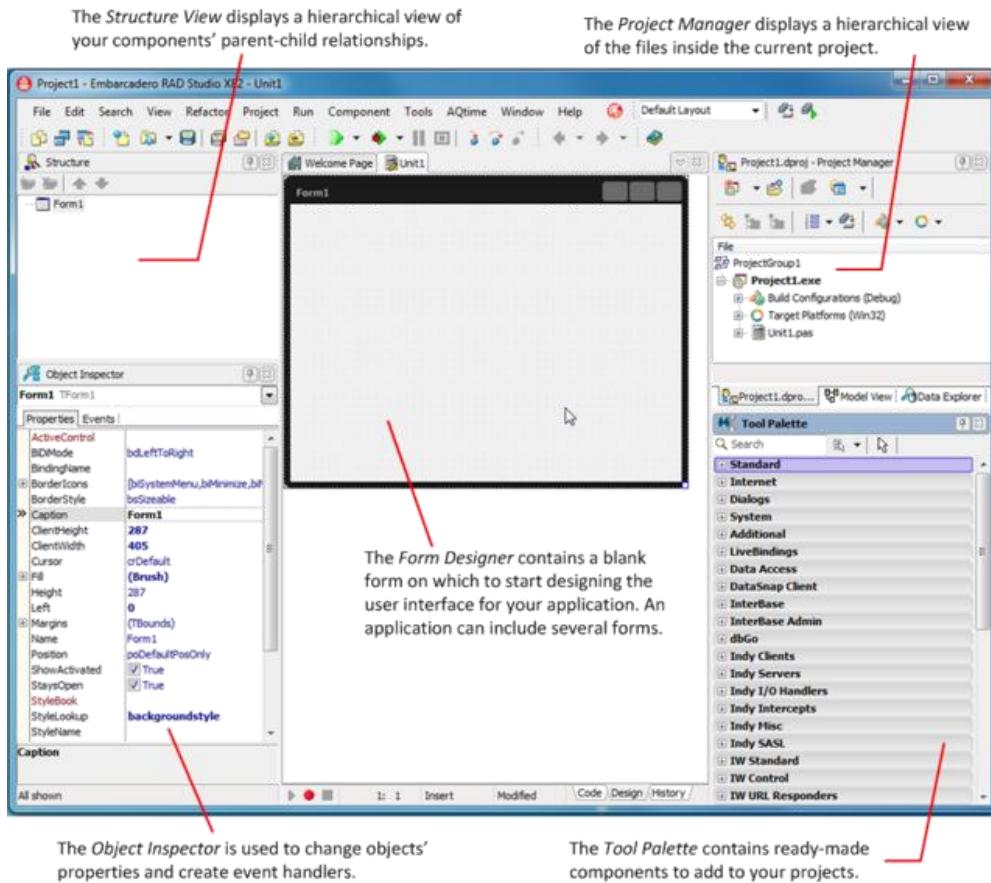
In lessons 3 you will explore the IDE in much more detail. Topic areas for lesson 3 include:

- The Main Menu and Toolbars
- The View menu

E-Learning Series: Getting Started with Windows and Mac Development

- Creating your own custom IDE layouts
- Customizing the **File > New** menu choices
- The Welcome Page and Creating Favorite Project Groups
- Using Project Templates from the Object Repository
- Customizing the main form of your application using the Object Inspector and Code Editor
- Adding Components to your Application Using the Form Designer and Tool Palette
- Customizing components using the Object Inspector, Property Editors and Component Editors
- Code Editor and History Lists
- The Structure View, Delphi Class Explorer, C++ Class Explorer
- Project Manager, Build Configurations and Option Sets
- Compiling and Running the Application
- Debugging the Application

When you start 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, and view and manage your application in various ways.



Put on your gloves, don your hard hat, turn on your halogen head lamp and let's go IDE spelunking!

The Main Menu and Toolbars

The IDE's Main Menu contains all of the main commands for creating projects, managing projects, compiling applications, debugging projects, creating components, setting IDE options, quick selecting any open windows, help/documentation and the desktop layouts toolbar.



Just below the Main Menu is the Toolbar area which contains speed buttons for commonly used IDE functions including View Unit, View Form, Toggle between Unit and Form, file operations, and run/debug commands.

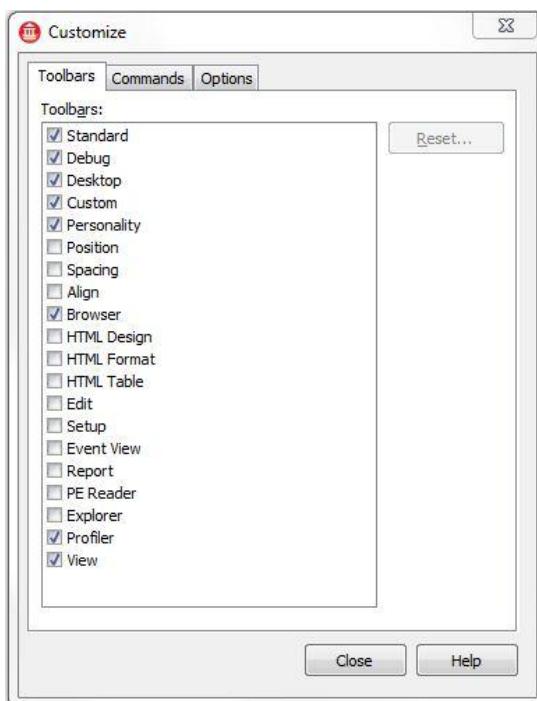
Toolbars can also be placed in multiple rows (the picture above shows my toolbar all on one row). The **View > Toolbars** menu item allows you to choose the toolbars that are displayed in the IDE. If you right-click an empty area on the Toolbar (located below the menus at the top of the IDE).

To arrange your toolbars

1. Click the grab bar on the left side of any toolbar.
2. Drag the toolbar to another location or onto your desktop.

To add icons to the toolbar

1. Choose View > Toolbars > Customize...
2. Click the Commands tab.
3. In the Categories list, select a category to view its tool icons.
4. From the Commands list, drag the selected icon onto the toolbar of your choice.
5. When completed, click Close.

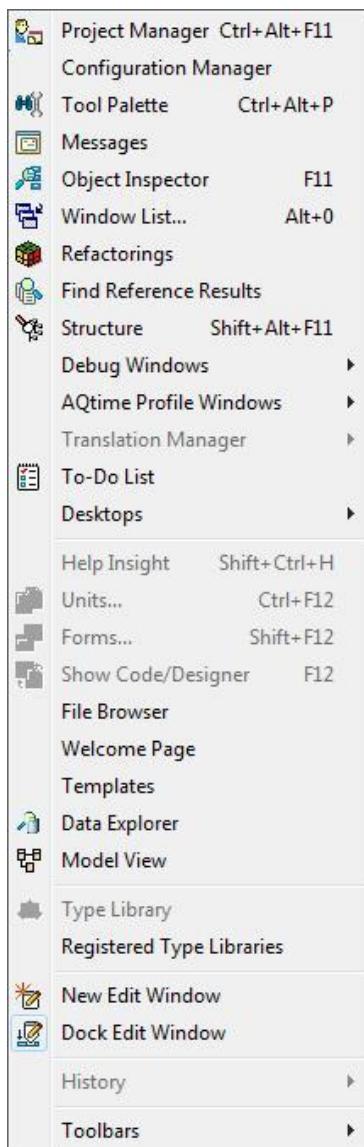


To delete icons from the toolbar

1. Choose View > Toolbars > Customize...
2. From the toolbar, not the Customize dialog box, drag the tool from the toolbar until its icon displays an X and then release the mouse button.
3. When completed, click Close.

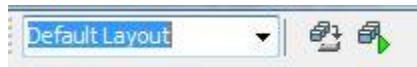
The View menu

The commands and dialog boxes on the **View** menu invoke managers, windows, browsers, and other tools for viewing information in RAD Studio. There are hot keys for many of the views. You can see the hot keys to the right of the menu items.



Creating your own custom IDE layouts

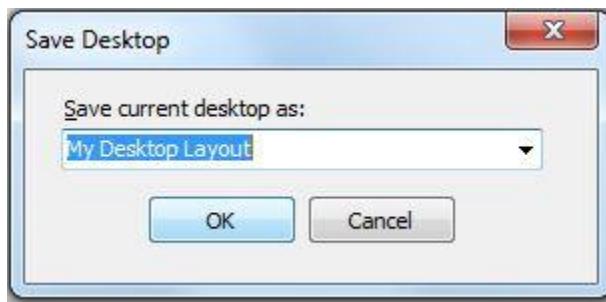
There are several default IDE layouts that you can use. You can choose a preset desktop layout using the **View > Desktops** menu item or by clicking on the down arrow in the desktop layout toolbar on the right side of the Main Menu.



The Desktops toolbar allows you to choose among the preset desktop layouts and any custom desktop layouts you have saved and to also save your current layout and set the layout you want to use for debugging. Desktop layouts can be used to create and manage windows. Your choices when using the **View > Desktops** menu item include:

- None - Does not specify a preset desktop layout.
- Classic Undocked - Emulates earlier Delphi versions, with separate windows for the menus and palette, designer, etc.
- Debug Layout - Customized for debugging, with call stack, thread, and other views shown instead of the default windows used for designing applications.
- Default Layout - Shows all windows docked into one container, with the most-used designing windows shown, including the tool palette, object inspector, design form, etc.
- Save Desktop - Invokes the Save Desktop dialog box.
- Delete - Invokes the Delete Saved Desktop dialog box.
- Set Debug Desktop - Invokes the Select Debug Desktop dialog box.

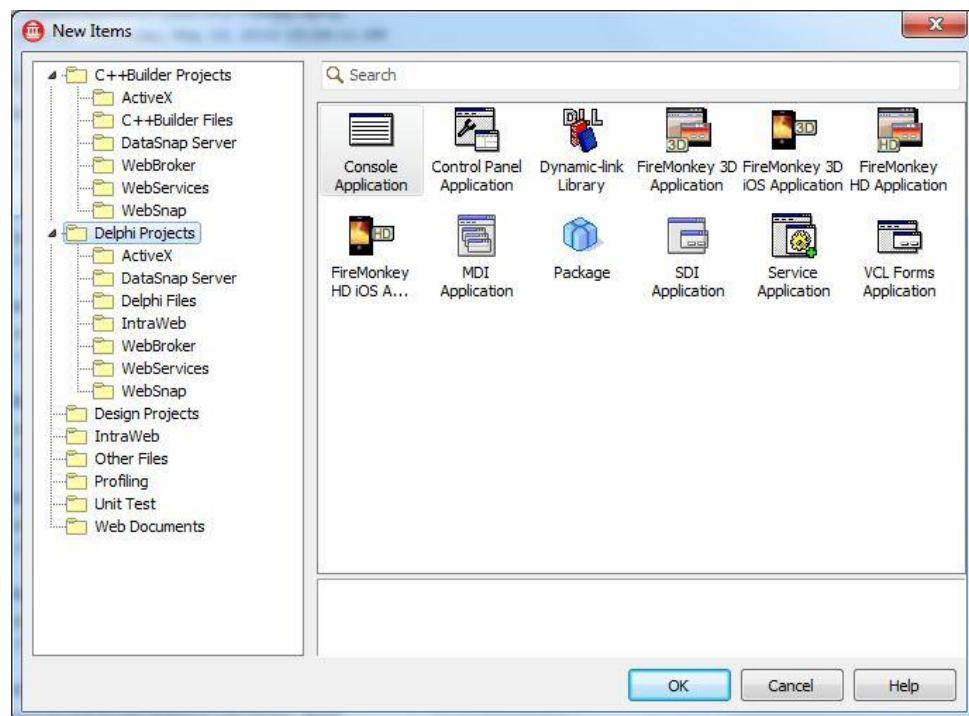
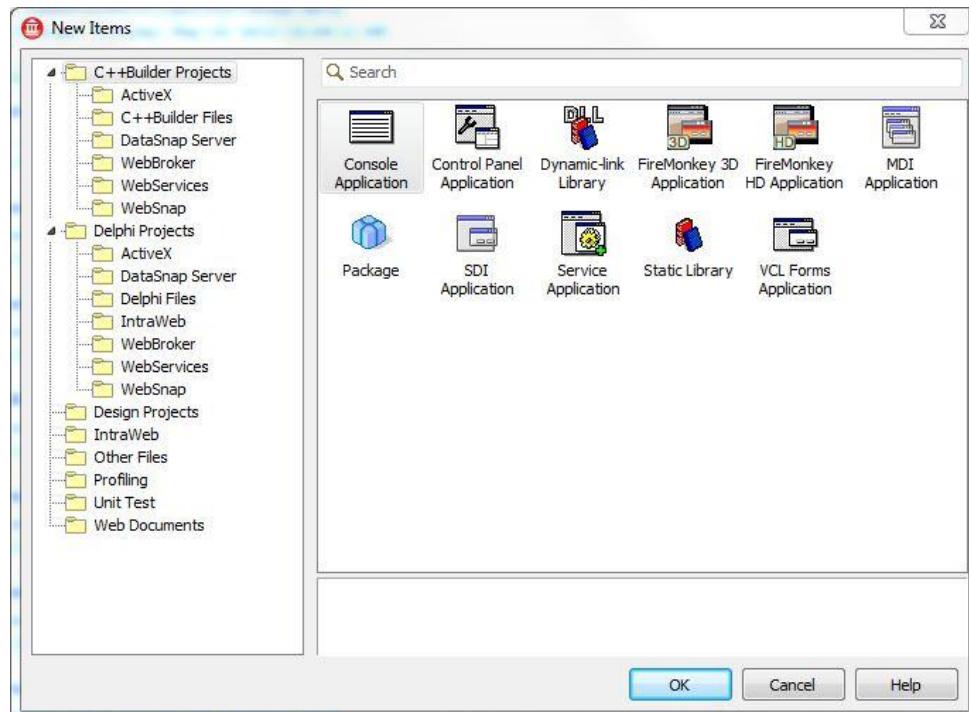
To save a custom desktop layout, configure the IDE the way you want and then click on the Save Desktop button or choose **View > Desktops > Save Desktop**. The Save Current Desktop dialog will appear and you can type in a new desktop name (or replace an existing named desktop)



Note: The current desktop layout is one of the items saved to a .dsk file when you set Autosave Project desktop on **Tools > Options > Environment Options**. The next time you open the project, the saved desktop and other settings are restored, including breakpoints, watches, and open files.

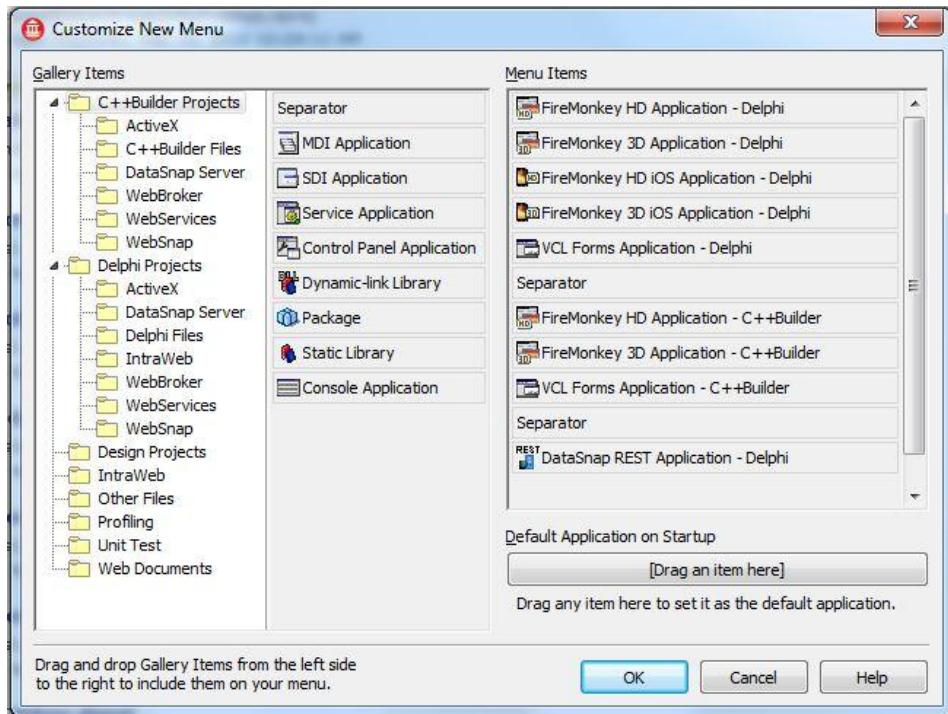
Customizing the File > New menu

After you install RAD Studio, the **File > New** menu will contain some of the common project types you can create. You can see a complete list of “New Items” by using the **File > Other...** menu item.



E-Learning Series: Getting Started with Windows and Mac Development

You can also customize your **File > New** menu to contain the project and file types that you use the most often. Customize the **File > New** menu by choosing the **File > New > Customize...** menu item.



Use this dialog box to customize the content of the **File > New** menu by dragging menu items from the center pane and dropping them on the right pane. There are three areas in the Customize New Menu dialog box:

- **Gallery Items** - displays the folders of gallery items that are available in the Object Repository. Click a folder to display its content in the center pane. The first item in the center pane is the "Separator" which is used to add separator line(s) in the **File > New** menu items.
- **Menu Items** - displays the items that are currently listed on the **File > New** menu. To remove an item from the **File > New** menu, drag it away from the list until its icon displays an X, and release the mouse button. To change the text for a menu item, double-click the text and enter new text. To add a separator bar between menu items, drag the Separator item from the center pane to the menu list.
- **Default Application on Startup** - if you want to set a default application type, drag the item that represents the application type from the center pane and drop it on this button. To remove the default application, click the button.

When you are finished customizing your File > New menu, click the OK button.

The Welcome Page and Creating Favorite Project Groups

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

E-Learning Series: Getting Started with Windows and Mac Development

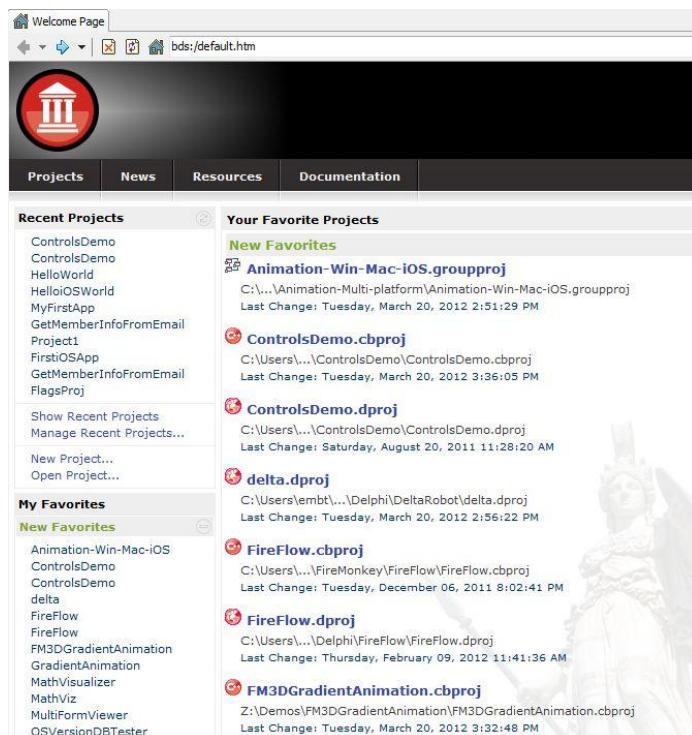
access them from the list of recent projects at the top of the page. If you close the Welcome Page, you can reopen it by choosing **View > Welcome Page**.

The welcome page has its own menu bar where you can select recent and new projects, news from the Embarcadero blogs and developer network, information about partner tools, Embarcadero web sites and product documentation. You can use the home icon to get back to the Welcome page and use the arrow icons to move back and forth for different pages (just like you would use in a browser).

The Welcome Page displays the recent projects you've been working on.



You can make any of your recent projects a Favorite by clicking on the “Make me a Favorite” link to the right of the project. You can also create “favorite” groups of projects to quickly load them with a single click. I usually have my Welcome Page set to show my favorite projects.



E-Learning Series: Getting Started with Windows and Mac Development

To create a favorite project group, click on the “Manage Favorites” link on the left hand side of the Welcome Page. At the bottom of the “Manage your Favorite Projects” page, give your favorites group a new name and click the “Create” link.

The screenshot shows a web-based application interface for managing favorite projects. At the top, there's a header bar with the title "Manage Your Favorite Projects". Below it, a section titled "New Favorites" lists several project files: Animation-Win-Mac-iOS.groupproj, ControlsDemo.cbproj, ControlsDemo.dproj, delta.dproj, FireFlow.cbproj, FireFlow.dproj, FM3DGradientAnimation.cbproj, GradientAnimation.dproj, MathVisualizer.dproj, MathViz.cbproj, MultiFormViewer.dproj, OSVersionDBTester.dproj, OSVersionInfo.cbproj, ShaderFilters.dproj, Text3DEditor.dproj, TrackMix.dproj, Wave.cbproj, and Wave.dproj. Each item has "move" and "delete" links next to it. Below this list is another section titled "Create New Favorites Group" with a text input field containing "My Favorites" and a "create" button.

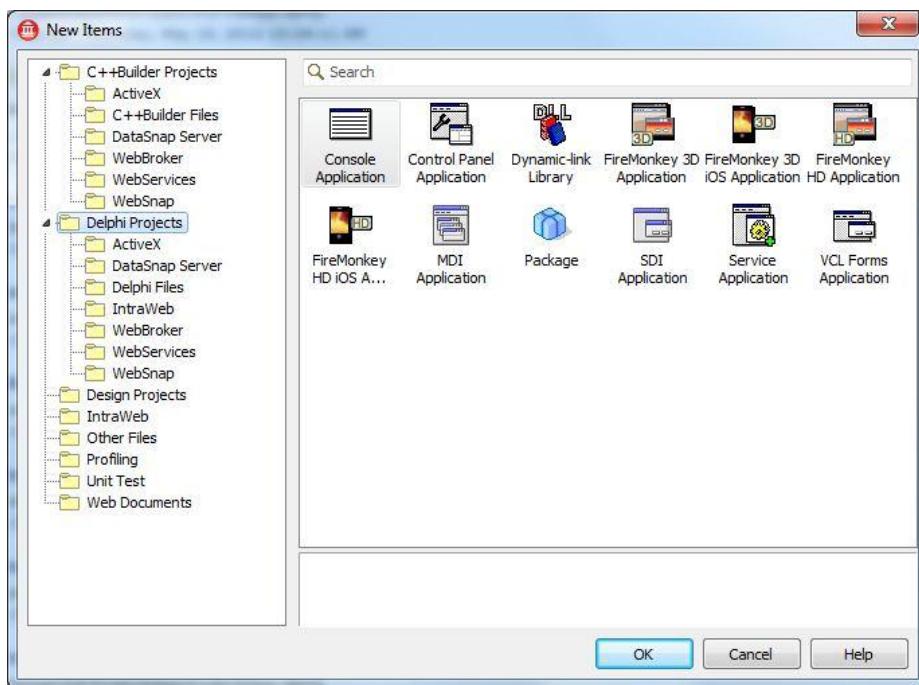
After you have created a favorites group, you can move any of your new favorites into the group by clicking on the “move” link to the left of a project. In the pull down list box select the favorite group for the selected project and it will be moved to that list.

This screenshot shows the same "Manage Your Favorite Projects" page after a project has been moved. The "New Favorites" section now contains the same list of projects as before. However, the "move to:" dropdown menu at the bottom of the page is open, showing "New Favorites" as the selected option. Other options in the dropdown include "My Favorites" and "MyFirstApp.dproj".

You can also click the “delete” link to remove any projects from a favorite project list.

Using Project Templates from the Object Repository

The **Object Repository or Gallery** (**File > New > Other...**) makes it easy to create new projects, forms, dialog boxes and other project and file types. It also provides templates for new projects and wizards that guide the user through the creation of forms and projects.



The Object Repository:

- Is maintained in the file **RADStudioRepository.xml** (located in the \ObjRepos directory), an XML file that contains references to the items that appear in the New Items dialog.
- Is filtered by platform so that the list contains only those project or form types that are supported on the current target platform.

In the search box you can enter a search string: any part of the name of a project type or file type that you want to locate in the Gallery. The New Items dialog box automatically displays only those items that match the string as you type. Click X to clear the search string. Two well-known wild cards are supported in the search:

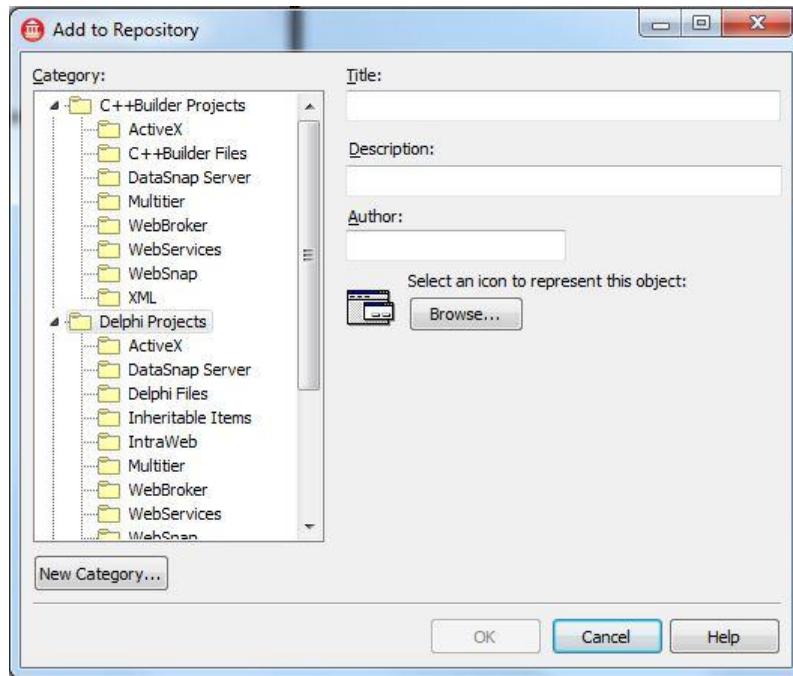
- * matches as many characters as possible.
- ? matches any single character.

An inherent * is added to the beginning of each search string that you type, and to the end as well, to allow matching of sub strings.

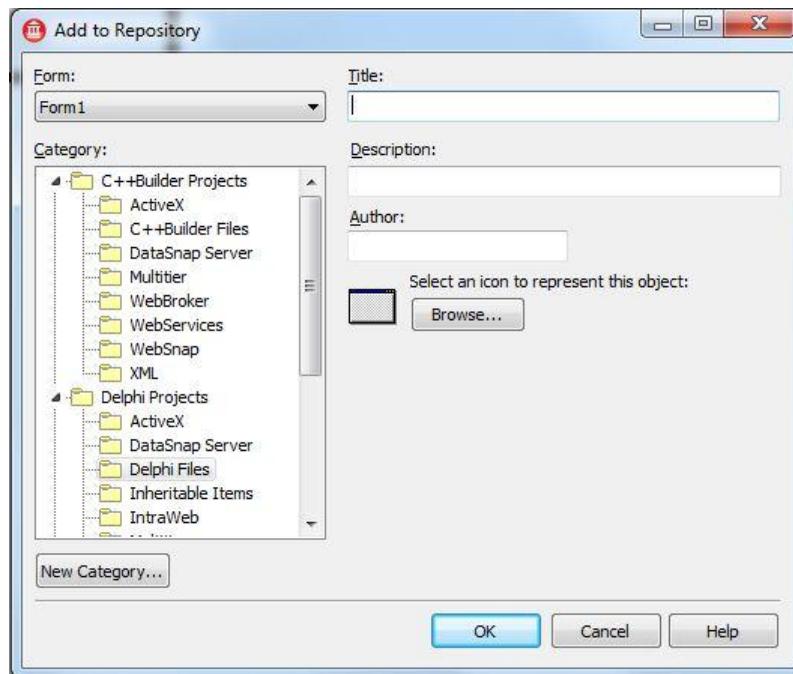
If you hover the mouse over an item, in the Object Repository, a description of the item (if available) will be displayed in the help/comments area.

E-Learning Series: Getting Started with Windows and Mac Development

You can add your own projects, forms, and other project files to those already available in the Object Repository. If the item is a project or is in a project, open the project. For a project, choose the **Project > Add to Repository...** menu item.



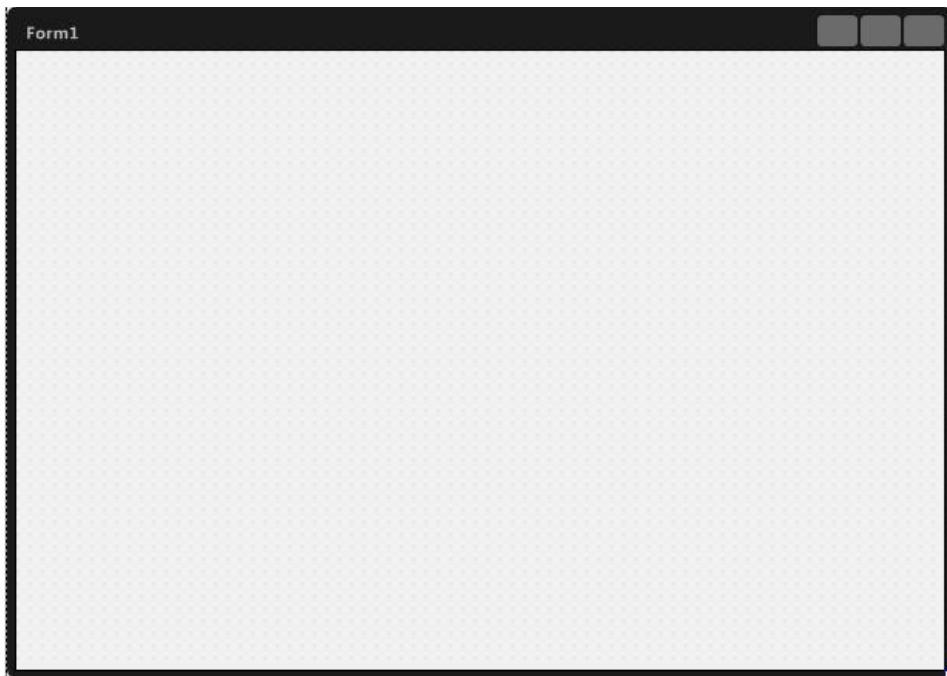
For a form or data module, right-click in the item and choose **Add to Repository...**



For both, type in a title, description, and author. Select an icon to represent the object by clicking the “Browse...” button. Decide in which category you want the item to appear or you can click on the “New Category...” button and type a new category name. When you are finished, click the OK button.

Customizing the Main Form of your Application using the Object Inspector and Code Editor

Using the **File > New** menu, select the “FireMonkey HD Application – Delphi” or “FireMonkey HD Application – C++” project wizard. A project will be created with a main program and a main form that is empty.

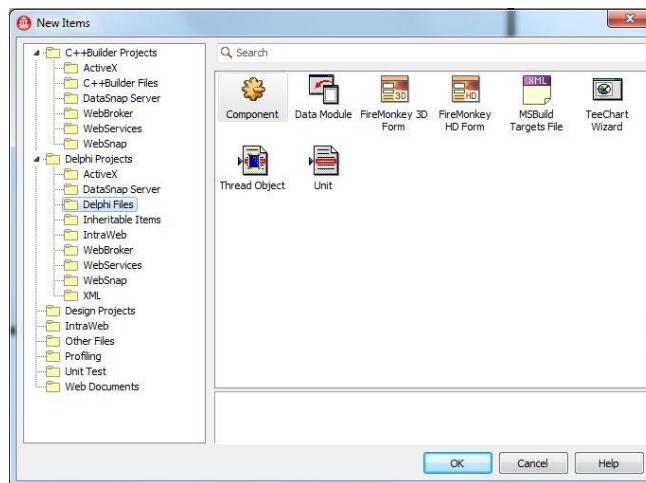


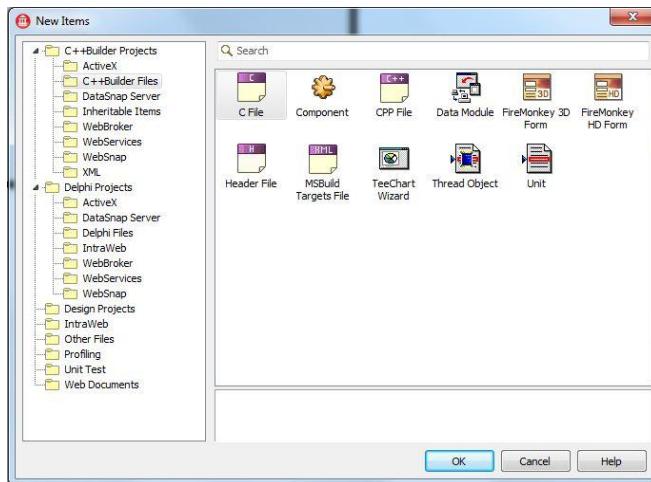
Use the Object Inspector to change the form caption to “My Main Form”. You can set the size of the main form in the Object Inspector by changing the ClientHeight and ClientWidth properties. You can also change the main form size by clicking the left mouse button on the lower right hand corner of the form and dragging the mouse to visually set the form size. In the Object Inspector you can set the WindowState property to wsNormal, wsMinimized and wsMaximized. There are additional properties that you can set in the Object Inspector.

E-Learning Series: Getting Started with Windows and Mac Development

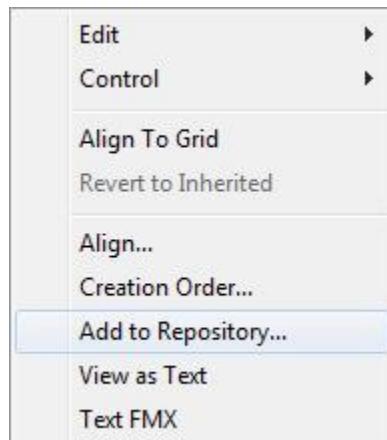


You can also add additional forms to your project using the **File > New > Other...** then choose “C++Builder Projects | C++Builder Files | FireMonkey HD Form” or Delphi Projects | Delphi Files | FireMonkey HD Form”.





Another way to customize the main form (or any other form) is by using the Code Editor. To view a form as text, right mouse click on the form and choose "View as Text" from the popup menu.



The code editor will appear instead of the form designer. The text view of the form will contain the object for the form and any properties that have values that are different from their default values.

```
object Form1: TForm1
  Left = 0
  Top = 0
  Caption = 'My Main Form'
  ClientHeight = 457
  ClientWidth = 674
  Visible = False
  WindowState = wsMaximized
  StyleLookup = 'backgroundstyle'
end
```

A screenshot of the code editor in Delphi. The code shown is the Delphi code for a TForm1 form. It defines the form's properties such as Left, Top, Caption, ClientHeight, ClientWidth, Visible, WindowState, and StyleLookup.

You can change the properties of the form using the code editor following the syntax `property = value`. Be careful when you are editing the form in code editor as there is no syntax checking. When you are finished, right-mouse click and choose “View as Form” (or use the Alt-F12 hot key) to return to the form designer to visually see the changes you made. If you’ve made a mistake in the form code editor you will see the following error:



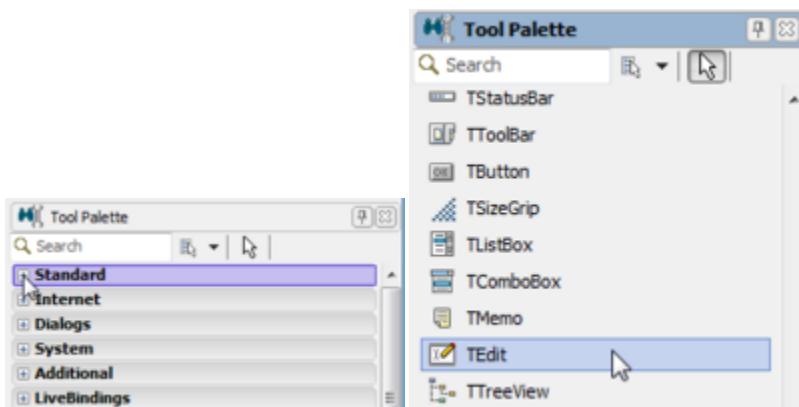
Click the Cancel button and fix any errors and then choose “View as Form”.

Adding Components to your Application Using the Form Designer and Tool Palette

In lesson 2 when you built your first Windows and Mac desktop application you saw how to find and add components to a form. There are hundreds of reusable components available in the IDE for creating user interfaces. There are three ways to find the components you want to use in your application.

Finding Components using the Tool Palette

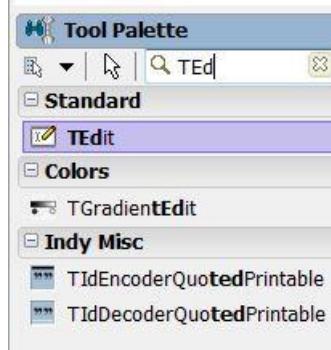
Move your cursor over the Tool Palette and expand the Standard category by clicking the plus (+) icon. Then select the **TEdit** component and drag and drop it onto the Form Designer (or double click on the TEdit component in the Tool Palette and it will be added to the form).



Finding Components using the Tool Palette’s Search Box

E-Learning Series: Getting Started with Windows and Mac Development

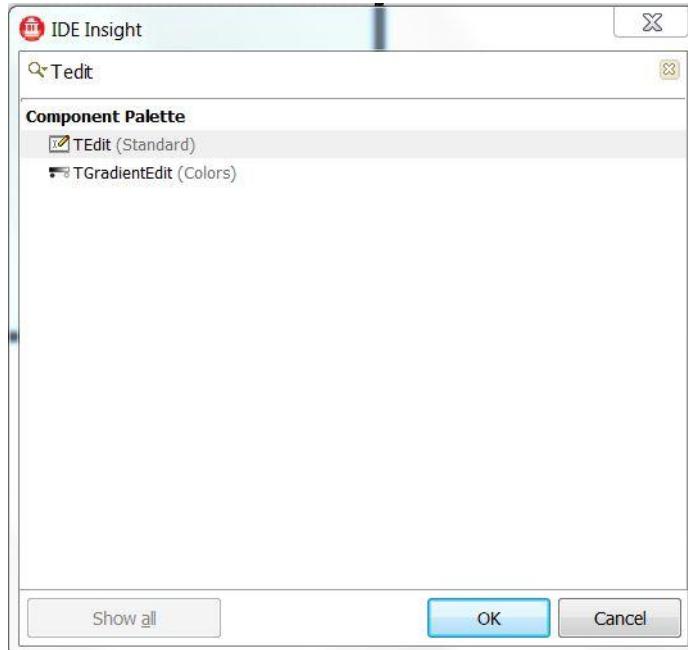
You can also use the Tool Palette's search box to find a component. Click in the search box and start typing the letters “ted” and then select the **TEdit** component from the component choices presented.



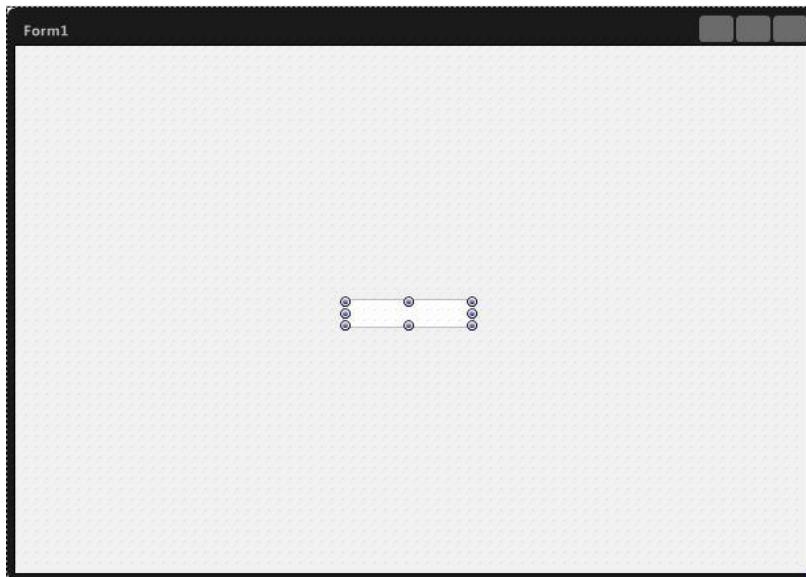
Then select the **TEdit** component and drag and drop it onto the Form Designer (or double click on the TEdit component in the Tool Palette and it will be added to the form).

Finding Components using IDE Insight

The third way to find a component is to use **IDE Insight** by hitting the F6 key (or Control .) and start typing the word “TEdit” and you’ll see several choices (Components, Templates, Projects and other IDE capabilities that match the text you are typing) start to appear. Select the TEdit component from the list to add it to the form.



Regardless of which method you use to find the component, an instance of the **TEdit** component will be displayed on the form.

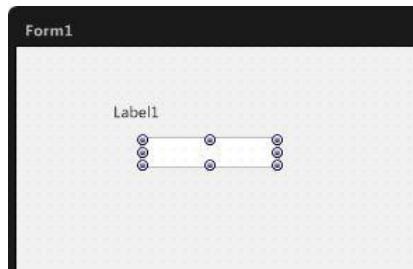


Compositing Components

Just as you can put multiple components on a form (the form is the parent container for the components), with FireMonkey every component can contain other components. This is called compositing components. For example, you can attach a TLabel component to the edit box. Use one of the methods above to put a TLabel on the form. Then in the Structure view, drag the TLabel component onto the TEdit component.



On the form designer, move the Label above and to the right of the edit box. Then use the mouse to move the edit box on the form. Notice that the label stays above and to the right of the edit box.



You can do this for any FireMonkey component. You can “parent” any number of components to another component.

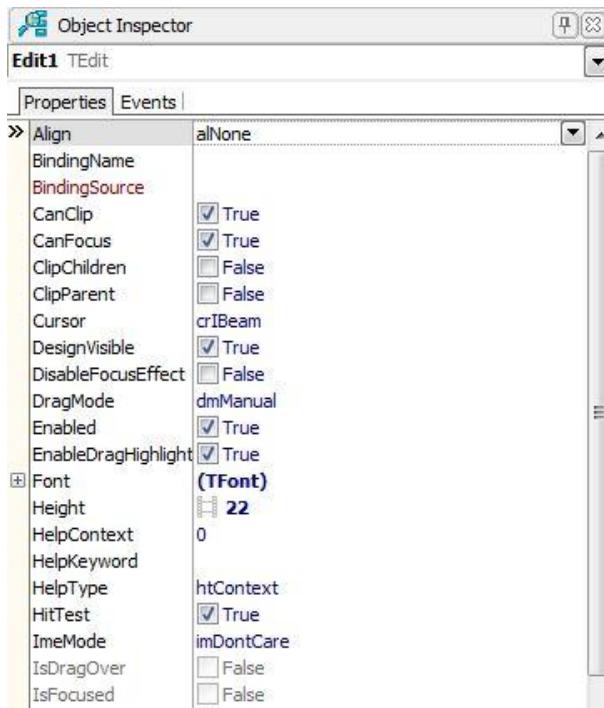
Customizing Components using the Object Inspector, Property Editors and Component Editors

Using the Object Inspector

The Object Inspector is the connection between your application's visual appearance and the code that makes your application run. The Object Inspector enables you to:

- Set design-time properties for components you have placed on a form (or for the form itself).
- Create and help you navigate through event handlers.
- Filter visible properties and events.

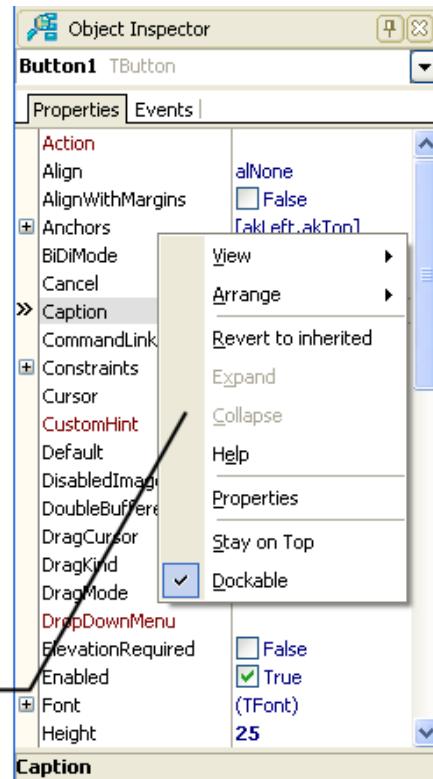
When you have your components placed on a form, you can use the Object Inspector to change any of the “published” properties. You can also change published properties in your program code. Components can also have Public and Private properties that don’t appear in the Object Inspector and can only be changed using code.



Use the Object Inspector to set the values you need for each component. To select a component you can:

- Click on a component in the Form Designer
- Select a component in the Structure View
- Use the **Instance List** at the top of the Object Inspector

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



Object Inspector context menu commands include:

- View - Filters the display of properties or events.
- Arrange - Sorts the property or events by name or by category.
- Revert to Inherited - Changes the property setting back to its original, inherited value.
- Expand - Expands the selected property or event.
- Collapse - Collapses the selected property or event.
- Help - Displays this Help topic.
- Properties - Displays the Object Inspector Properties dialog box, allowing you to change the appearance of the Object Inspector.
- Stay on Top - Displays the Object Inspector on top of the desktop even if other windows are displayed.
- Dockable - Enables drag-and-dock for the Object Inspector.

The Properties page displays the properties of the component that is selected on the form. If the Properties are arranged by name (the default), the first column on the Property page lists the names of the selected component's published properties as follows:

- If a plus sign (+) appears beside a property name, you can click the (+) to display the sub-properties of that property. The list can include the possible values when the property represents a set of flags (the value column lists the set enclosed in square brackets []), or sub-

properties if the property represents an object (the value column gives the name of the object, enclosed in parentheses).

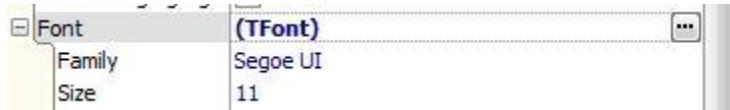
- Similarly, if a minus sign (-) appears, you can click the (-) to collapse the sub-properties. When a property has focus, you can also use the keyboard + and – keys to expand or collapse properties.

The second column on the Property page displays the property values as follows:

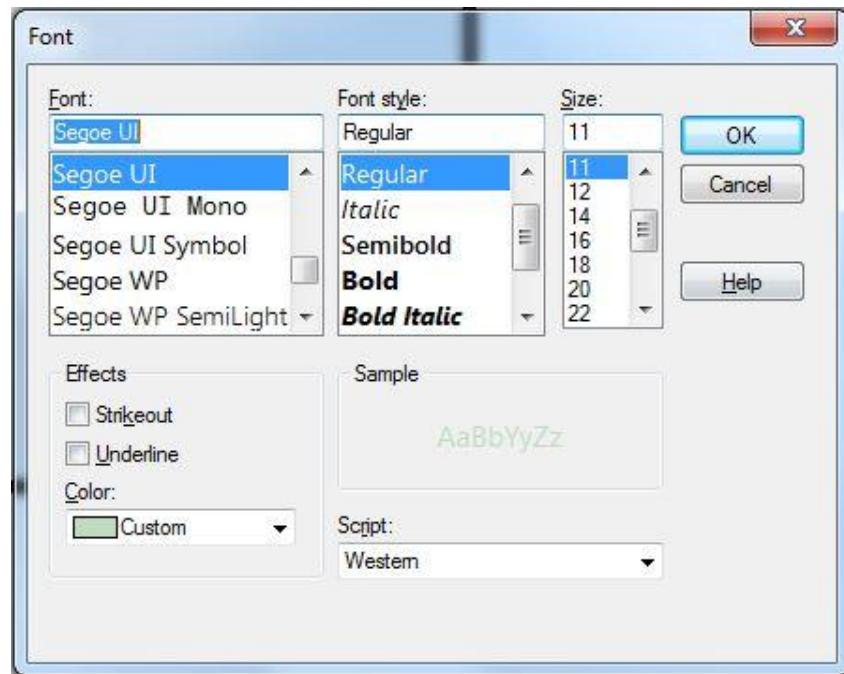
- When the property is selected, the value changes to an edit control where you can type a new value.
- If the value can be set using a dialog, an  ellipsis button appears when the property is selected. Click this button to display a dialog where you can set the property. You can also display the dialog by double-clicking the value column.
- If the value is an enumerated type, a  drop-down button appears when the property is selected. Click this button to display a drop-down list that you can use to set the property. You can see images in the drop-down lists for properties that include images such as cursors, brush types, colors, and image lists. To view images referenced by the ImageIndex property, you need to set the property that holds the image list to the image list containing the images.
- If the value is another component, you can shift the Object Inspector's focus to that component by holding down the Ctrl key while double-clicking.

Property Editors

Some properties, such as *Font*, have special property editors. Such properties appear with an ellipsis mark (...) next to their value when the property is selected in the Object Inspector.



To open the property editor, double-click in the *Value* column, click the ellipsis, or type Ctrl+Enter when focus is on the property or its value. With some components, double-clicking the component on the form also opens a property editor.

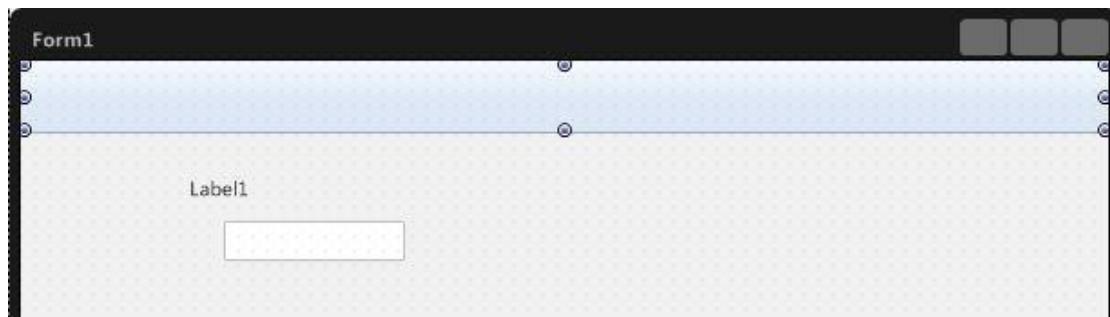


Property editors let you set complex properties from a single dialog box. They provide input validation and often let you preview the results of an assignment.

Component Editors

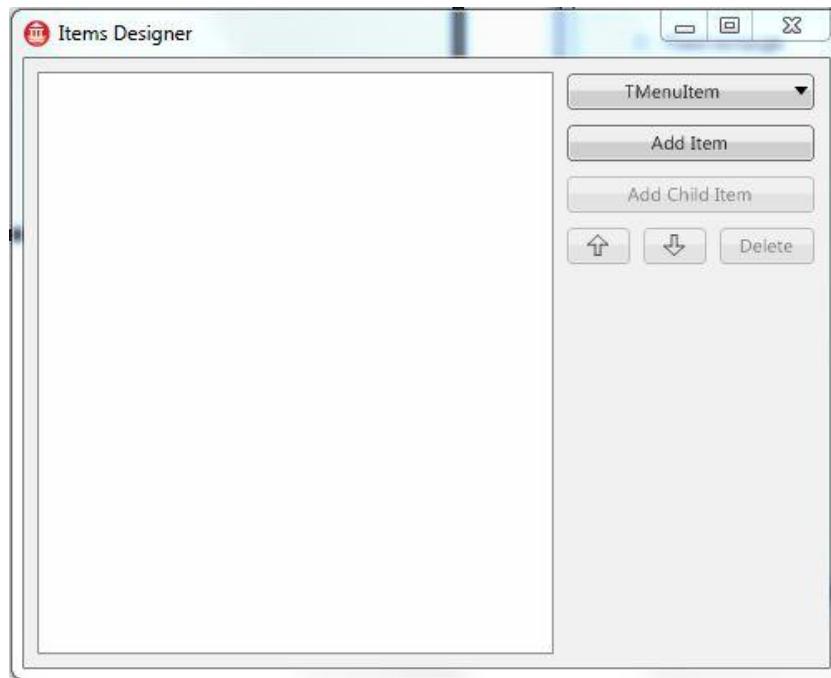
Component editors determine what happens when the component is double-clicked in the designer and add commands to the context menu that appears when the component is right-clicked. They can also copy your component to the Windows clipboard in custom formats.

TMenuBar is an example of a FireMonkey component that has a component editor. Add a **TMenuBar** component to your form. Set the Align property to **alTop**.

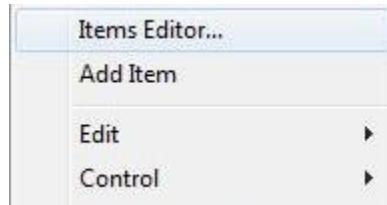


There are three ways to select one of **TMenuBar**'s component editors:

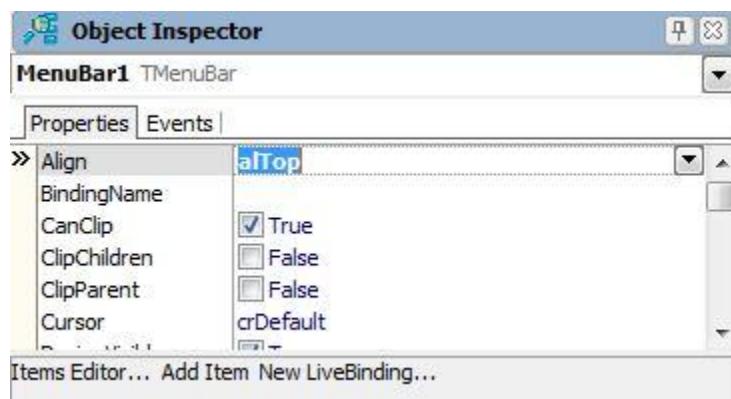
- 1) Double-Click on the **TMenuBar** in the Form Designer to bring up its default component editor.



2) Right Mouse Click on the TMenuBar in the Form Designer and select **Items Editor...** or **Add Item** from the top of the popup menu.



3) Click on the Items Editor... or Add Item entries at the bottom of the Object Inspector



Component editors are created by the component creator. As part of the component implementation, the component creator can add items to the context menu.

Code Editor and History Lists

The Code Editor

The Code Editor occupies the IDE's center pane (along with the Form Designer). The Code Editor is a full-featured, customizable UTF8 editor that provides syntax highlighting, multiple undo capability, and context-sensitive help for language elements.

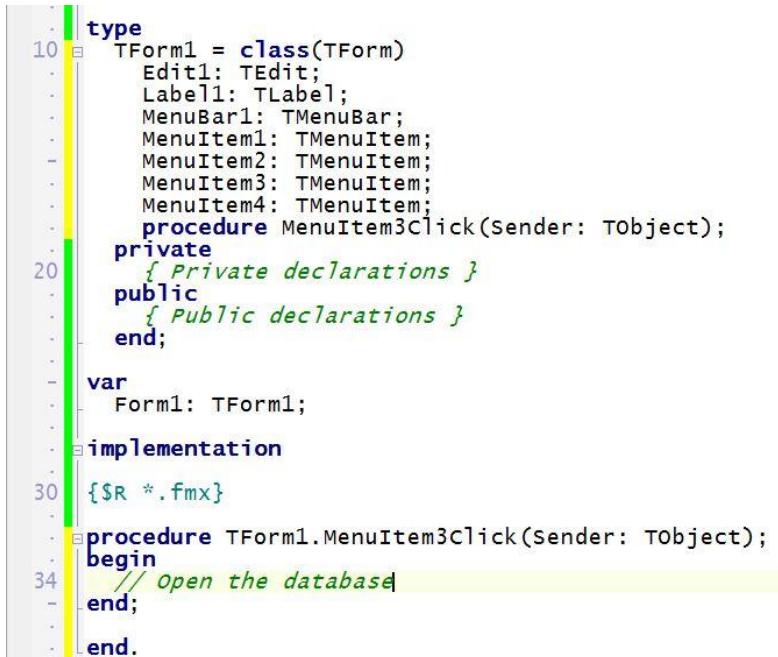
As you design the user interface for your application, RAD Studio generates the underlying code. When you modify object properties, your changes are automatically reflected in the source files.

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

To help you write code, the Code Editor provides many features including Change Bars, Indenting Code, Formatting Code, Code Insight, Code Parameter Hints, Code Hints, Help Insight, Code Completion, Class Completion, Block Completion, Code Browsing, Code Navigation, Live Code Templates, Code Folding, Refactoring, Sync Edit, To-Do Lists, Keystroke Macros, Bookmarks and Block Comments.

Change Bars

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



```

10  type
11    TForm1 = class(TForm)
12      Edit1: TEdit;
13      Label1: TLabel;
14      MenuBar1: TMenuBar;
15      MenuItem1: TMenuItem;
16      MenuItem2: TMenuItem;
17      MenuItem3: TMenuItem;
18      MenuItem4: TMenuItem;
19      procedure MenuItem3Click(Sender: TObject);
20  private
21    { Private declarations }
22  public
23    { Public declarations }
24  end;

25  var
26    Form1: TForm1;
27
28  implementation
29
30  {$R *.fmx}
31
32  procedure TForm1.MenuItem3Click(Sender: TObject);
33  begin
34    // open the database
35  end;
36
37 end.

```

The screenshot shows a code editor window with a vertical margin on the left. The margin contains line numbers (10, 20, 30, 34) and color-coded change bars. A green bar is at line 10, indicating no changes. A yellow bar is at line 34, indicating changes made since the last save. The main code area contains Delphi-style code for a form with menu items and a database connection.

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.

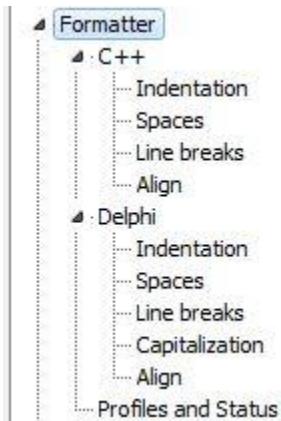
Indenting Code

You can use the TAB key to indent the current line of code or a block of code in the Code Editor. The number of spaces that the TAB key indents is determined by the Block indent option on the **Tools > Options > Editor Options > Source Options** dialog box.

- To indent a line of code, place the cursor at the beginning of the line and press TAB.
- To indent an entire block of code, highlight the code block and press TAB.
- To move text to the left ("outdent"), use SHIFT + TAB.

Formatting Code

RAD Studio provides the customizable source code formatter. Editing Delphi or C++ code in the Code Editor, you can apply the Format Source context menu command (or the **Edit > Format Source** menu command) to format the source code. You can set the Indentation, Spaces, Line Breaks, Capitalization, and Align formatting options under the Formatter group in the Options dialog box (**Tools > Options > Formatter**).



Notice that the Format Source command implements automatic formatting of your code. It takes into account only formatting options specified in the Options dialog box and totally overwrites all your manual formatting implemented in the Code Editor. For example, it ignores your manual code indenting. You can select a block of code and call the Format Source command, so that only this block will be formatted.

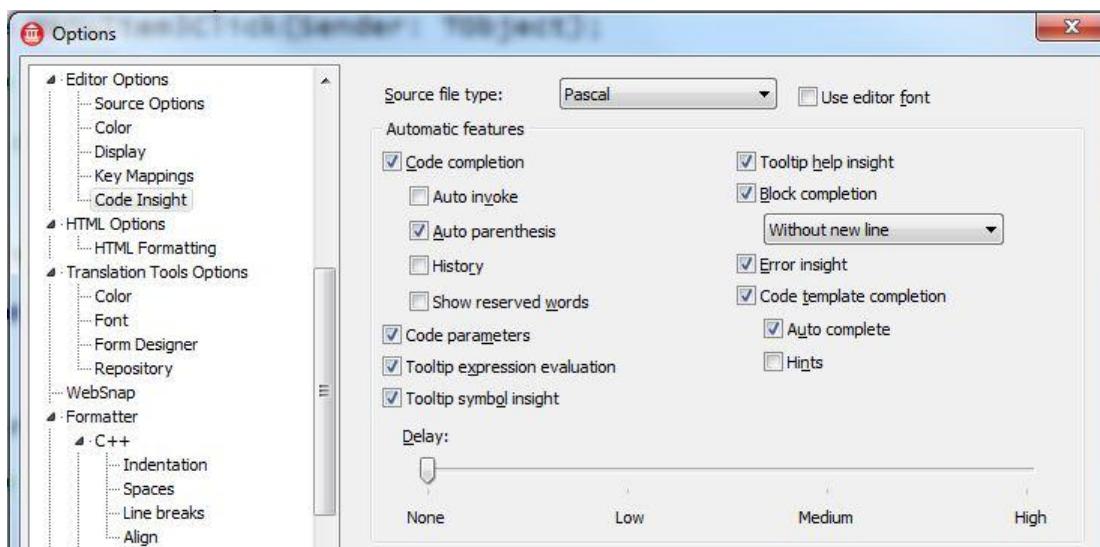
Note: The Editor Options pages of Tools Options provide additional code formatting options, including Source Options, Color, Display, Key Mappings, and Code Insight. See Customizing the Code Editor.

Code Insight

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 invoke Code Insight, press CTRL+SPACE while using the Code Editor. A pop-up window displays a list of symbols that are valid at the cursor location.

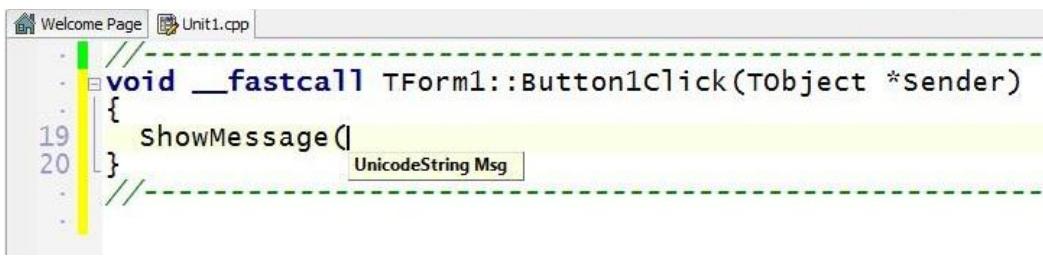
To enable and configure Code Insight features, choose **Tools > Options > Editor Options** and click Code Insight.



When you're using the Delphi Language, the pop-up window filters out all interface method declarations that are referred to by property read or write clauses. The window displays only properties and stand-alone methods declared in the interface type.

Code Parameter Hints

After you type the opening parenthesis for a method, function or procedure, the code editor displays a hint containing argument names and types between the parenthesis of a call, for example, `ShowMessage (/);`. You can invoke Code Parameter Hints by pressing CTRL+SHIFT+SPACE.



A screenshot of a C++ code editor showing a tooltip for the variable 'Msg'. The tooltip displays the type 'UnicodeString' and the declaration 'ShowMessage(UnicodeString Msg)'. The code is part of a TForm1::Button1Click event handler.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ShowMessage( UnicodeString Msg );
}
```

(C++)



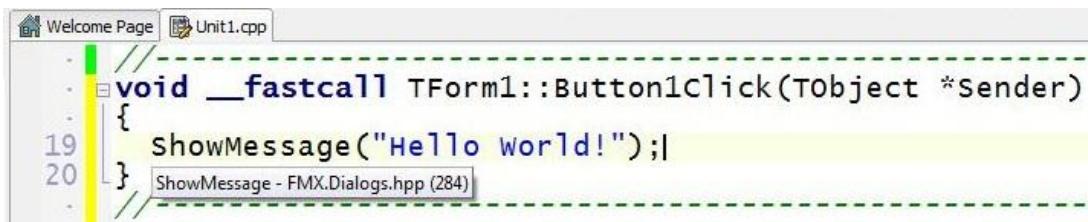
A screenshot of a Delphi code editor showing a tooltip for the variable 'Msg'. The tooltip displays the type 'string' and the declaration 'ShowMessage(const Msg: string)'. The code is part of a TForm1.Button1Click event handler.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ShowMessage( const Msg: string );
end;
end.
```

(Delphi)

Code Hints

If you hover the mouse over symbols in the editor, a hint will be displayed containing information about the symbol such as type, file, and line # declared at.



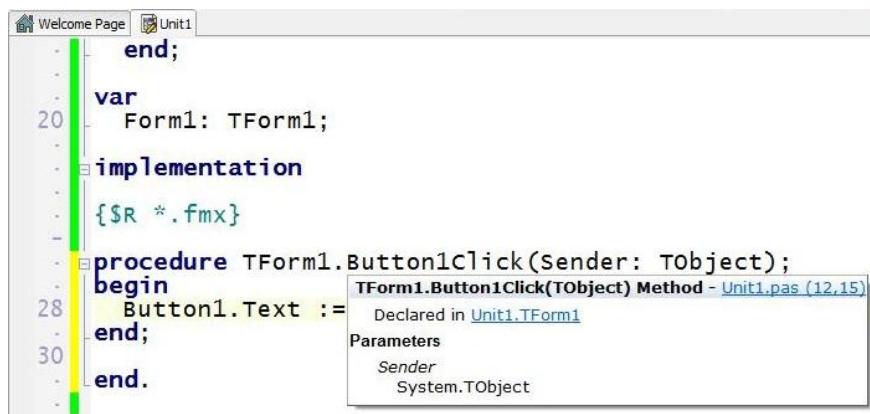
A screenshot of a Delphi code editor showing a tooltip for the 'ShowMessage' function. The tooltip displays the file 'FMX.Dialogs.hpp' and line number '284'. The code is part of a TForm1::Button1Click event handler.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ShowMessage("Hello world!");
}
```

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

Help Insight

Help Insight displays a hint containing information about the symbol such as type, file, line # declared at, 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.



Code Completion

The Code Completion feature displays a drop-down list of available symbols at the current cursor location. You invoke Code Completion for your specific language in the following way:

In Delphi:

- Press CTRL+SPACE (always invokes Code Completion).
- Enter . (only works when Auto Invoke is enabled on the Code Insight page).

In C++:

- Press CTRL+SPACE (always invokes Code Completion).
- Enter . or -> (only works when Auto Invoke is enabled on the Code Insight page).

To cancel a Code Completion request, press the ESC key.

Class Completion

Class completion simplifies the process of defining and implementing new classes by generating skeleton code for the class members that you declare. By positioning the cursor within a class declaration in the interface section of a unit and pressing CTRL+SHIFT+C (or right-clicking and selecting Complete class at cursor on the Code Editor Context menu), any unfinished property declarations are completed. For any methods that require an implementation, empty methods are added to the implementation section. You can also use class completion to fill in interface declarations for methods that you define in the implementation section.

Block Completion

When you press ENTER while working in the Code Editor and there is a block of code that is incorrectly closed, the Code Editor enters the closing block token at the next available empty line after the current

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 now have: begin end. This feature also works for the C++ language.

Code Browsing (Ctrl-Click)

While using the Code Editor to edit an application, you can use CTRL-click to automatically "jump to" the code that defines an identifier. To browse code, hold down the CTRL key while passing the mouse over the name of any class, variable, property, method, or other identifier. After the mouse pointer turns into a hand and the identifier is highlighted and underlined, click the highlighted 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 from the context menu. Code browsing can find and open only the units that exist in the project Search path or in the global Browsing path.

Directories are searched in the following order:

- Either the project-specific Search path for Delphi (**Project > Options > Delphi Compiler**) or the Include path for C++ (**Project > Options > Directories and Conditionals**).
- 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**).

Code Navigation

Code Navigation allows you to navigate your code while you are using the Code Editor. There are several Code Navigation capabilities including Method Hopping, Finding Classes, Finding Units, Finding the Next and Previous Changes and searching source code for usages (Delphi only).

Method Hopping

You can navigate between methods using a series of editor hotkeys. You can also lock the hopping to occur only within the methods of the current class. For example, if class lock is enabled and you are in a method of **TComponent**, then hopping is only available within the methods of **TComponent**.

The keyboard shortcuts for Method Hopping are as follows:

- **CTRL+Q^L** -- toggles class lock.
- **CTRL+ALT+UP** -- moves to the top of the current method, or the previous method.
- **CTRL+ALT+DOWN** -- moves to the next method.
- **CTRL+ALT+HOME** -- first method in source.
- **CTRL+ALT+END** -- last method in source.

- **CTRL+ALT+MOUSE_WHEEL** -- scrolls through methods.

Finding Classes

Use the **Search > Find Class** command to see a list of available classes that you can select. After you choose one, the IDE navigates to the class declaration.

Finding Units

If you are programming in the Delphi language, you can use a refactoring feature to locate namespaces or units. Use the Find Unit command to locate and add units to your code file.

Finding the Next and Previous Changes

As you edit code, you can use keystrokes to quickly navigate to the Next and the Previous changes that you have made. The keyboard shortcuts are:

- **Ctrl+Shift+F7** -- moves to the previous line modified since the file was opened (green marking in the gutter).
- **Ctrl+Shift+F8** -- moves to the next line modified since the file was opened (green marking in the gutter).
- **Alt+Shift+F7** -- moves to the previous line modified since the last save (yellow marking in the gutter).
- **Alt+Shift+F8** -- moves to the next line modified since the last save (yellow marking in the gutter).

If the next or previous line is in an elided (folded) block, the block is unfolded. Code folding is described later in this topic.

Keystrokes are the same for the following keyboard mappings: Default, IDE classic, Brief, and Epsilon. The Visual Studio and Visual Basic keyboard mappings do not have keystrokes for Next/Previous change.

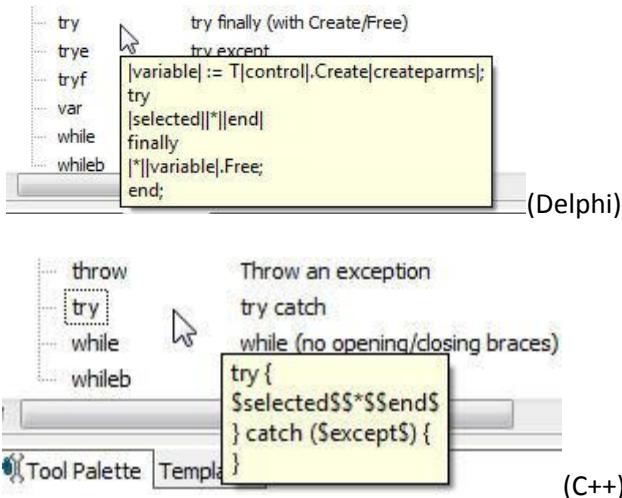
Searching Source Code for Usages

If you are programming in the Delphi language, you can use the Search for Usages feature to find usages of classes, methods, variables, and overriding methods, derived classes and implemented interfaces in your source code projects.

Live Code Templates

Live Templates allow you to have a dictionary of pre-written code that can be inserted into your programs while you're working with the Code Editor. This reduces the amount of typing that you must do.

To view the Live Code Templates available for your Delphi or C++ code, use the **View > Templates** menu to display the **Templates Window**. The Live Templates list will co-exist in the Tool Palette area of the main window as a separate tab.



You can insert one of these pre-defined code skeletons into your code by double-clicking the template name in the **Templates Window**. You can also type the template name and press the **Tab** key to insert the template. The XML files for the RAD Studio Live Templates are located in *C:\Program Files\Embarcadero\RAD Studio\9.0\ObjRepos\en\Code_Templates*.

You can add your own code templates to the **Templates Window**. The names of live templates can either represent the code in the template (such as *class*) or be the first word in the code (such as *try*), and some template names are close (but not exact) versions of words you might type (such as *forr*). Templates that you create (and templates provided by third party add-ins) are saved by default in the *\My Documents\RAD Studio\code templates* directory.

Additional Live Code Templates details are available on the Embarcadero DocWiki at http://docwiki.embarcadero.com/RADStudio/en/Live_Templates.

Code Folding

Code folding lets you collapse or expand regions or blocks of code. Collapsing your code creates a hierarchical view of the code and makes it easier to read and navigate. The collapsed code is not deleted, but hidden from view until you expand it. Code folding is on by default.

To use code folding, click the plus (+) and minus (-) signs located on the left edge of the Code Editor:

- Click the plus (+) sign to expand the associated region of code.

- Click the minus (-) sign to collapse the associated region of code.

```
unit Unit1;
interface
uses
  System.SysUtils, System.Types,
  System.UITypes, System.Classes,
  System.Variants,
  FMX.Types, FMX.Controls,
  FMX.Forms, FMX.Dialogs;
type
  TForm1 = class(TForm)
var
  Form1: TForm1;
implementation
```

To enable/disable code folding:

- Use the Code Folding check box on the Tools > Options > Editor Options dialog box.
- Use the keyboard shortcut Ctrl+Shift+K+O.

You can also use the context menu Fold command to fold specific types of regions, such as Types, Methods, XML Doc comments (for Delphi), and the 'Nearest' region. The Fold and Unfold commands are described in Code Editor Context Menu. For more information about code folding, including how to create code folding regions, see Using Code Folding.

Refactoring

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 method refactoring. 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.

You will find additional information about refactorings for Delphi and C++ on the Embarcadero DocWiki at http://docwiki.embarcadero.com/RADStudio/en/Refactoring_Overview and http://docwiki.embarcadero.com/RADStudio/en/Refactoring_Code.

Sync Edit

The Sync Edit feature lets you simultaneously edit identical identifiers in selected code. For example, in a procedure that contains three occurrences of label1, you can edit just the first occurrence and all the other occurrences will change automatically. To use Sync Edit in the Code Editor, select a block of code that contains identical identifiers.

Click the **Sync Edit Mode** icon  that appears in the left gutter. 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 the **TAB** key to move between each identifier in the selection.

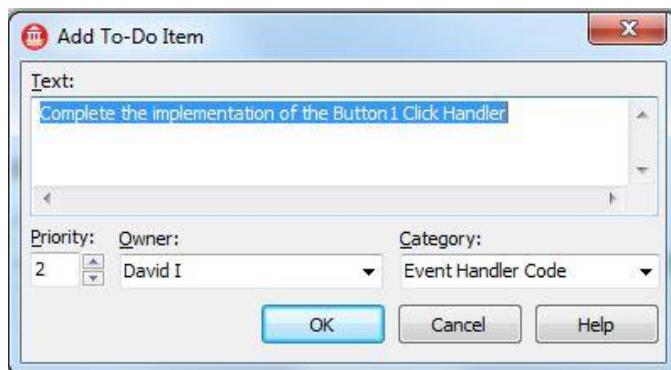
Begin editing the first identifier. As you change the identifier, the same change is performed automatically on the other identifiers. By default, the identifier is replaced. To change the identifier without replacing it, use the arrow keys before you begin typing.

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

Note: Sync Edit 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, Sync Edit is intended for small sections of code, such as a single method or a page of text. For changing larger sections of code, consider using refactoring

To-Do Lists

A To-Do List records 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. Set the cursor in the code editor where you want to place a To-Do List item. Right Mouse Click and choose “Add To-Do Item...” from the context menu.



After you click the **OK** button the To-Do List Item is added as a comment in your source code. You can view all of your To-Do List Items using the **View > To-Do List** menu item.

The screenshot shows the Delphi IDE interface. The Code Editor at the top contains the following Pascal code:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Text := '42';
  { TODO 2 -oDavid I -cEvent Handler code : Complete the imple
end;
```

Below the editor is a toolbar with icons for Run, Stop, and Break, followed by buttons for Insert, Modified, and tabs for Code, Design, and History. A status bar at the bottom shows "31: 45".

Underneath the toolbar is a "To-Do List - Project1" window. It has columns for Action Item, Status (!), Module, Owner, and Category. There is one item listed:

Action Item	Status	Module	Owner	Category
<input type="checkbox"/> Complete the implementation of the Button1 ...	!	2 C:\Users\em...\\Unit1.pas	David I	Event Handler Code

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

Keystroke Macros

You can record a series of keystrokes as a macro while editing code. 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.

Bookmarks

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

When you set a bookmark, a book icon is displayed in the left gutter of the Code Editor. You can use up to ten bookmarks, numbered 0 through 9, within a file.

You can drag-and-drop bookmark icons in the gutter of the Code Editor, and a moved bookmark has the number of the original bookmark.

Block Comments

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 prefixed with // and is ignored by the compiler. Pressing CTRL+// adds or removes the slashes, based on whether the first line of the code is prefixed with //. When using the Visual Studio or Visual Basic key mappings, use CTRL+K+C to add and remove comment slashes.

History List

The **History Manager** lets you see and compare versions of a file managed by the IDE, including:

- Multiple backup versions saved by the IDE

E-Learning Series: Getting Started with Windows and Mac Development

- Your saved local changes
- The current buffer of unsaved changes for the active file.

You can also see Subversion information if your project or individual files are under version control by Subversion. You can see version control information in the History Manager that originates from both the IDE and **Subversion**:

- If the current file is under version control, all types of revisions are available in the History Manager.
- If you are using Subversion, the Subversion Integration in the IDE provides the following functionality that you can use inside the IDE:
 - All the revisions that have been checked into the Subversion repository are available in the History Manager.
 - You can diff (compare) files and annotate specific versions of the file in the History Manager.
 - You can perform the standard source-control operations in the Project Manager, such as Add to repository, Update, and Commit.

To open the **History Manager**, click the **History** tab, which is located in the center of the lower edge of the IDE main window:



The History Manager contains three tabbed pages (Contents, Information, and Differences):



Contents - Displays the current and previous versions of the file selected in the dropdown list in the toolbar of the History window. The Contents page contains two panes:

- Revision content: The top portion of the Contents page lists revisions of the active file.
- Columns include:
 - Revision: Revision number of each revision of the active file
 - Label: Either the Revision number or a specified label
 - Date: The date of the revision
 - Author: The name of the user who checked in the revision
- File display: The lower portion of the Contents page displays the contents of the active file.
Note: To see the Subversion log entries for a specific file:
 - Hover the mouse over a revision (or) that is checked into Subversion.
 - Click the Information tab.

Information - For the selected revision of the active file, displays the following:

- The Label (either the revision number or a specified label such as "Local file")
- The Comments entered with the commit of that revision

Differences - Displays the differences between the selected versions of the active file. The Differences page contains three panes:

E-Learning Series: Getting Started with Windows and Mac Development

- Differences From: Lists the revisions of the active file, including the current revision.
- To: Lists the revisions of the active file, including the current revision and any unsaved changes in the buffer.
- File display: The lower portion of the Differences page displays the contents of the file being diff'ed. Highlighting indicates the differences between the revisions selected in "Differences From:" and the revisions selected in "To:".
 - (a large minus sign) marks source lines that were deleted.
 - (a large plus sign) marks source lines that were added.
- You can set the colors to be used in the highlighting on the Differences tab by using the **Tools > Options > Editor Options > Color** dialog box.

The screenshot shows the Delphi IDE's Differences view. At the top, there are two tables: 'Differences From:' and 'To:', both listing 'File' and 'Buffer' revisions with their respective dates. Below these tables is a code editor window displaying Pascal code. The code includes declarations for TForm1, TEdit, TLabel, TMenuBar, and several TMenuItem components. It defines procedures for MenuItem3Click and MenuItem5Click, and a var section with Form1. The implementation section contains code for MenuItem3Click, which opens a database, and MenuItem5Click, which edits some text. The code editor uses color-coded highlighting to indicate differences: green for additions and red for deletions. The status bar at the bottom indicates '3 differences found'.

```
10  TForm1 = class(TForm)
.    Edit1: TEdit;
.    Label1: TLabel;
.    MenuBar1: TMenuBar;
.    MenuItem1: TMenuItem;
.    MenuItem2: TMenuItem;
.    MenuItem3: TMenuItem;
.    MenuItem4: TMenuItem;
18 +| MenuItem5: TMenuItem;
.    procedure MenuItem3Click(Sender: TObject);
20 +| procedure MenuItem5Click(Sender: TObject);
.    private
.        { Private declarations }
.    public
.        { Public declarations }
.    end;
.
.    var
.        Form1: TForm1;
30  implementation
.
.    {$R *.fmx}
.
.    procedure TForm1.MenuItem3Click(Sender: TObject);
.    begin
.        // open the database
.    end;
.
.    40 +procedure TForm1.MenuItem5Click(Sender: TObject);
.    +begin
.        // edit some text
.    +end;
.    +
.    end.
```

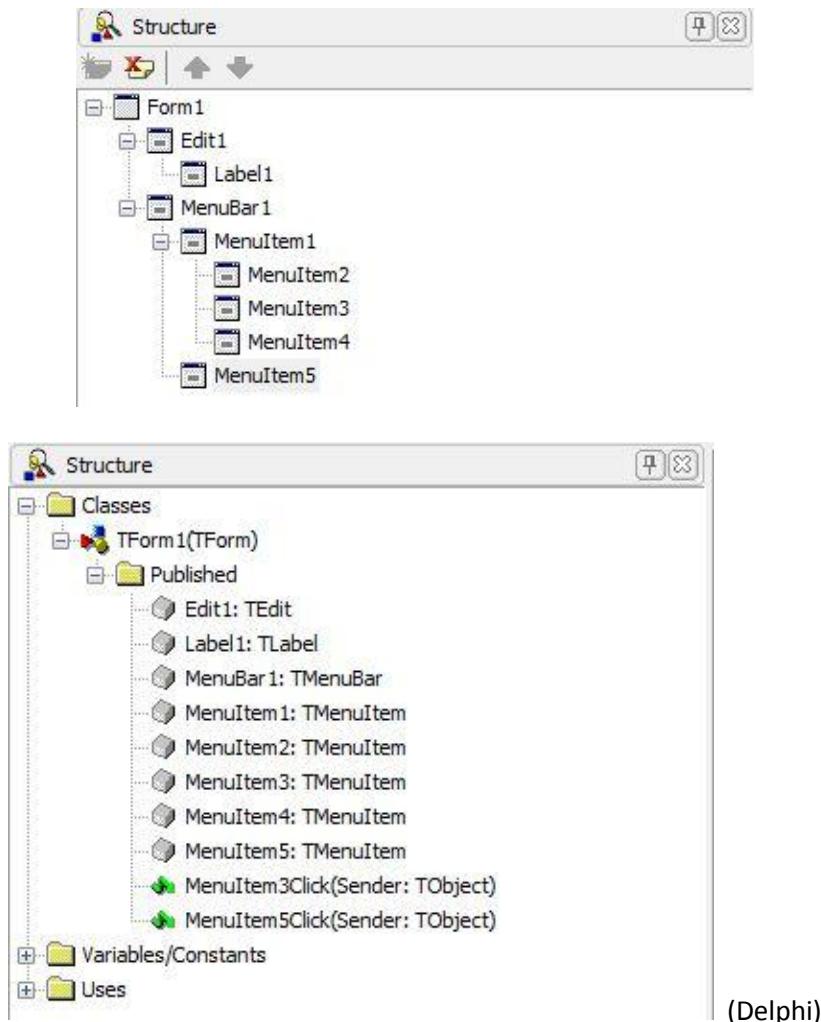
The Structure View, Delphi Class Explorer and C++ Class Explorer

The Structure View, Delphi Class Explorer and C++ Class Explorer let you visually see the components, classes and code used in your application. You can use them to display properties in the Object Inspector, show relationships and navigate to the declarations in the code editor.

Structure View

The Structure View displays the hierarchy of the following:

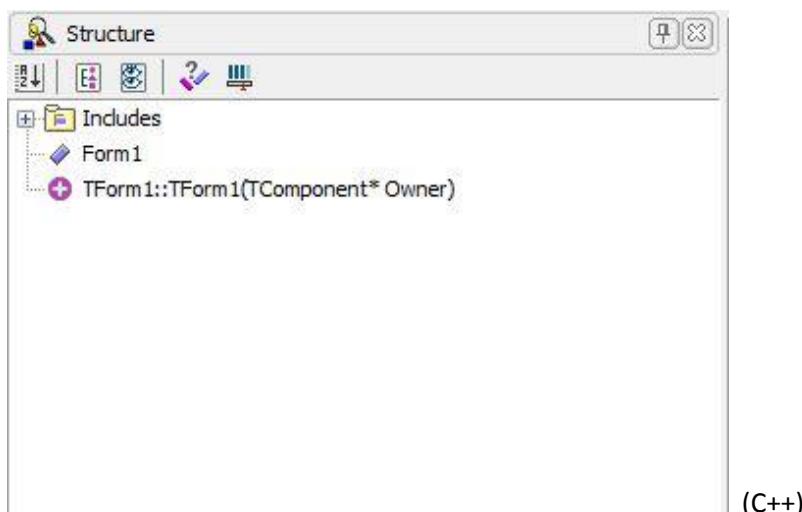
- Source code or HTML displayed in the **Code Editor**. When displaying the structure of source code or HTML, you can double-click an item to jump to its declaration or location in the **Code Editor**.
- Components displayed on the **Designer**. When displaying components, you can double-click a component to select it on the form.



The Structure view contains a toolbar for C++ application development that allows you to control how the contents of the Structure view are displayed. It consists of the following buttons (from left to right):

- Sort Alphabetically - Sorts the contents of the Structure view alphabetically.
- Group by type - Groups items into folders by type in the Structure view.

- Group by visibility - Groups class members into folders by visibility: public, protected, private, and published. For C++, 'Classes' is a generic group that encompasses classes, structs, unions and templates.
- Show function and variable type - Displays the type to the right of the member in the Structure view.
- Filter by visibility (private, protected, public, _published) - Toggles the Structure view display through four different visibility levels. You can selectively display several different combinations of elements that were declared with the access specifiers “_published”, “public”, “protected” and “private”.



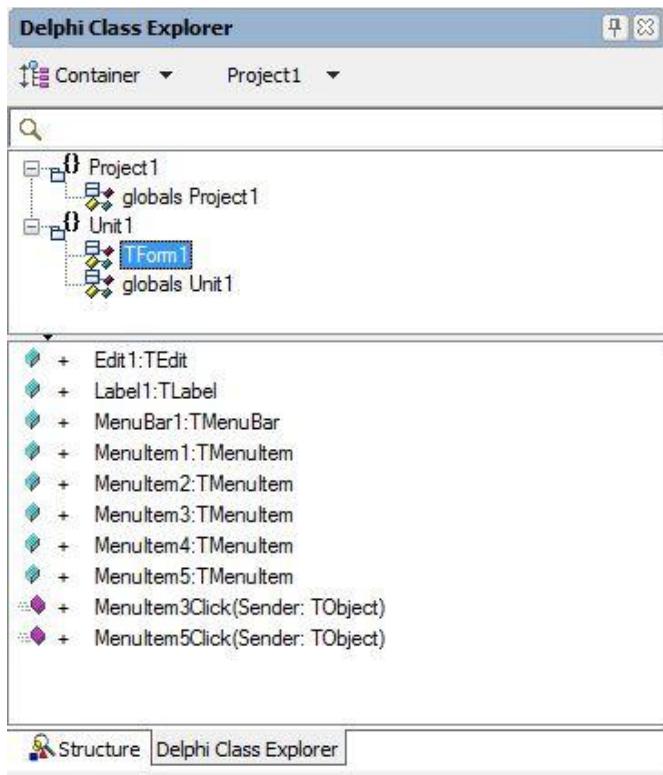
If your code contains syntax errors, the errors are displayed in the **Errors** node in the **Structure View**. Double-click an error to locate the corresponding source in the **Code Editor**. (Not applicable for C++ development.)

You can use the Structure View to view relationships between components and between database objects. For example, if you add a panel component and a check box component to your form, they are considered *siblings*. But if you use the Structure View and drag the check box on top of the panel icon, the check box becomes the *child* of the panel component.

If you double-click a Designer component in the Structure View, the Code Editor opens to a place where you can write an **event handler** for the component. You can control the content and appearance of the **Structure View** by choosing **Tools > Options > Environment Options > Explorer** and changing the settings.

Delphi Class Explorer

The **Delphi Class Explorer** makes it easy to navigate through your Delphi project unit files, viewing the hierarchical structure of declared types, classes, interfaces, and namespaces. The **Delphi Class Explorer** also automates the creation of members (fields, methods, properties).



The **Delphi Class Explorer** window has three areas:

- The **Search** control occupies the upper line of the **Delphi Class Explorer** window. It contains the lens icon.
- The **Class View** pane occupies the central part of the **Delphi Class Explorer** window.
- The **Member List** pane.

The **Search** control provides quick location of entities in the **Class View** pane. Type a string and all entities in the **Class View** pane whose names match this string become highlighted. The first matched entity becomes selected.

The **Class View** shows a tree structure of types, classes, interfaces, and namespaces declared directly in your project.

The **Member List** displays local and global members (fields, properties, and methods) declared in a node (class or interface) selected in the **Class View**.

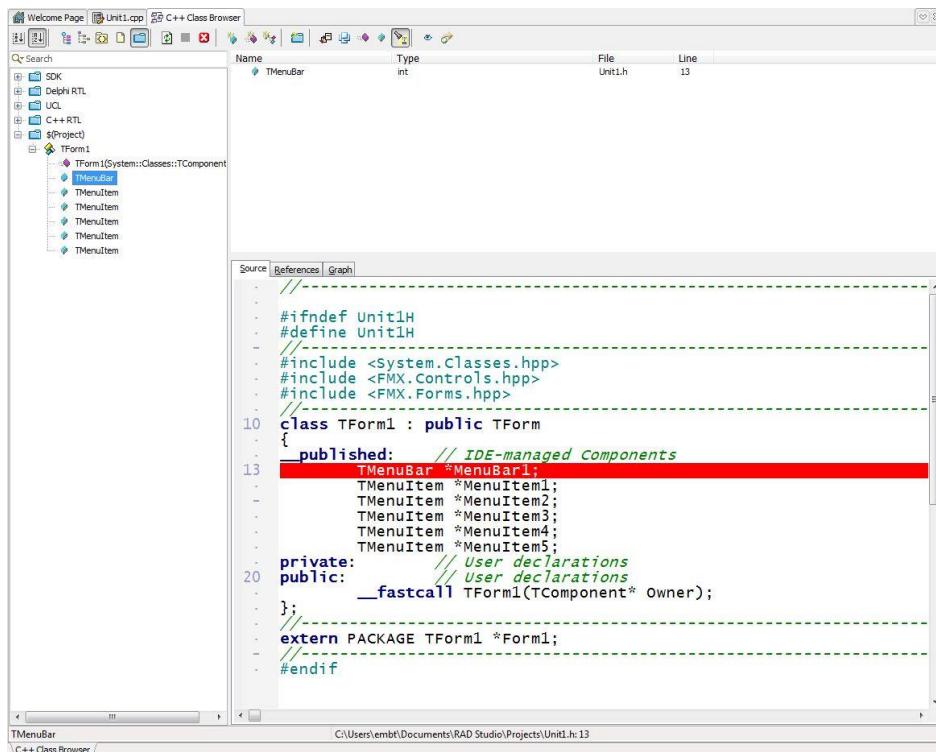
To open the **Delphi Class Explorer** window, click the **View > Delphi Class Explorer** menu item. By default, the **Delphi Class Explorer** window appears docked to the upper-left corner with the **Structure View**.

The **Delphi Class Explorer** parses all unit files registered in an open project. The **Delphi Class Explorer** uses the obtained information to order nodes (units, classes, interfaces, types) in the **Class View** tree and to order members (fields, properties, and methods) displayed in the **Member List**.

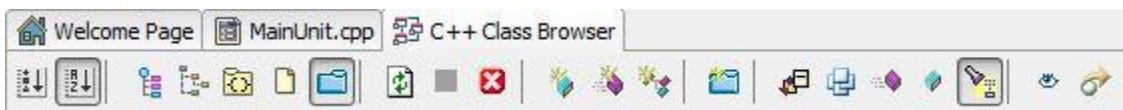
C++ Class Explorer

The C++ Class Explorer allows you to:

- Examine the class structure of your project
- Examine the declaration in the source for a selected element
- Create new fields, methods, and properties in managed units in your project



The C++ Class Explorer window includes a toolbar with buttons.



These buttons allow you to (left to right):

- Sort by types, either forward or reverse. Sorts the various types in the following order: Classes, Interfaces, Structs, Enumerations, Typedefs, Methods and Variables. The reverse button sorts individual types in the reverse order. In either case, the Type List displays the types in alphabetical order (the default view).
- Sort alphabetically by name; forward (A-Z). The reverse button sorts backward (Z-A).
- Do not group types. Types are shown in a flat display, as root nodes, instead of being associated with parent nodes (namespace, file, or Custom File Group nodes).
- Group types by inheritance hierarchy. Types are shown under their base type nodes; or flat, if there's no base type.

E-Learning Series: Getting Started with Windows and Mac Development

- Group types by their namespaces. Each node is represented by a namespace icon, and the types in the namespace are listed under that icon.
- Group types by their files. Displays a file icon () for each file (.cpp, .h, or .hpp file), and lists the elements defined in each file.
- Group types using custom file groups. Uses the default display groups, as well as any custom groups that you have created.
- Refresh list of types. Parse modified source files and update type list.
- Cancel active browser compilations.
- Clear all browser information
- Add a new method to this class - opens the Add Method (C++) dialog box. Enabled only for managed units.
- Add a new field to this class - opens the Add Field (C++) dialog box. Enabled only for managed units.
- Add a new property to this class - opens the Add Property (C++) dialog box. Enabled only for managed units.
- Configure custom file groups - opens the Explorer File Groups dialog box, enabling you to: Create your own file groups in the Type List, Enable/disable the display of specific groups, and establish the display order of the groups in the Type List.
- Show top-level typedefs in the type list. Default is Off.
- Show top-level enumerations in the type list. Default is On (shown).
- Show top-level functions in the type list. Default is Off.
- Show top-level (global) variables in the type list. Default is Off.
- Include inherited members. By default, the Member View includes inherited class members by displaying the members of the ancestor type below the members of the selected type. For example, TForm3 is followed by its ancestor type, TForm. When this button is not pressed, only locally defined members are displayed. Default is On.
- Go to the declaration of the selected node. Opens the source in the Code Editor, and highlights the declaration of the selected item.
- Go to the definition of this declaration. Opens the source in the Code Editor, and highlights the definition of the selected item. The C++ compiler maintains information such as when each symbol is first defined and when it is subsequently used. This information is used to perform the Go to definition command.

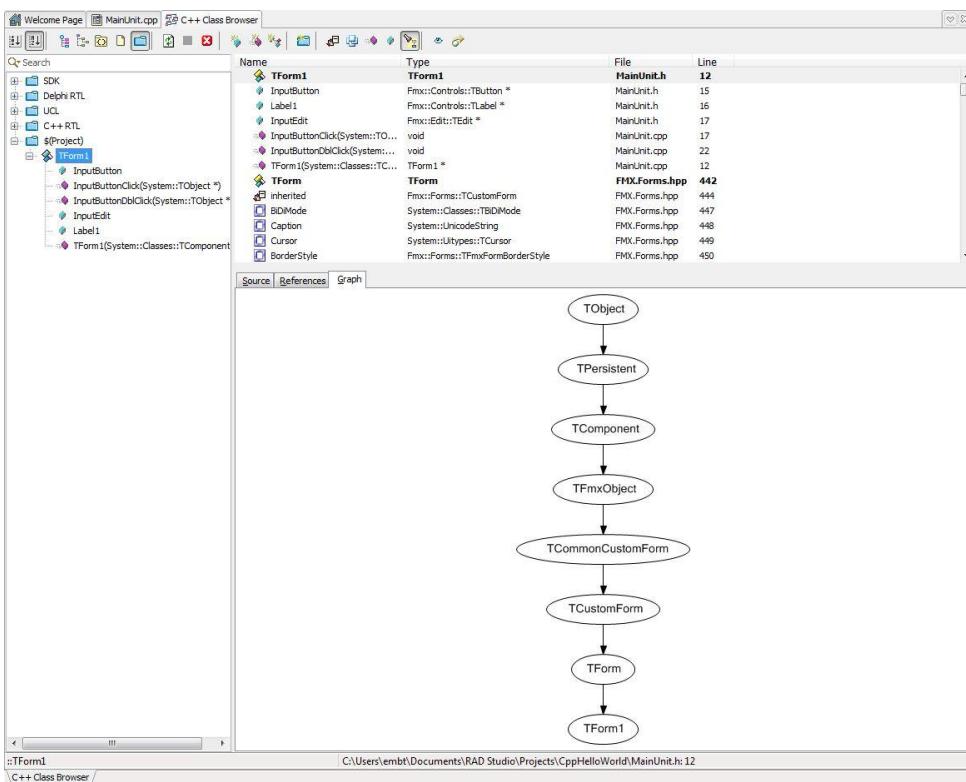
At the top of the Type List is an incremental search field, indicated by a magnifying glass icon. As you enter a string in the search field, the Type List adjusts to display only elements that contain that string. The search matches the names of any contained elements (such as members of a class), even if the class name itself does not contain the string.

The C++ Class Explorer window is separated into three panes:

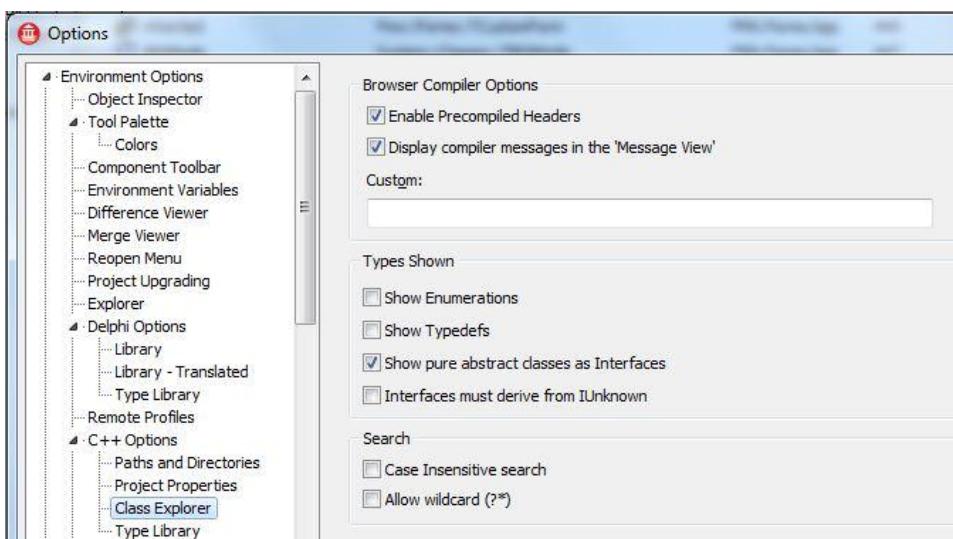
- Type List - Shows a tree structure of types, classes, interfaces, and namespaces declared directly in your project.
- Member View - Displays the members (fields, properties, and methods) of the type or types currently selected in the Type List.
- Source/References/Graph Window - The three tabs display different information, as follows:
 - Source tab displays the source code, and focuses on the declaration of the item most recently selected in the Type List.

E-Learning Series: Getting Started with Windows and Mac Development

- References tab lists the references to the symbols currently selected in the Type List. Double-clicking an entry in the References tab opens the Code Editor window and navigates to the reference.
- Graph tab displays graphical information about the items currently selected in the Type List. For example, if you select a few classes in the Type List, the Graph tab displays the hierarchy of the selected symbols.



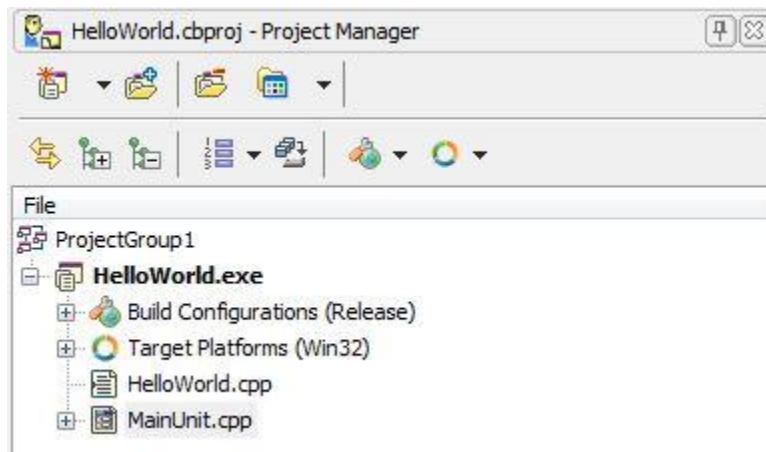
You can set options for the C++ Class Explorer by using the **Tools > Options > C++ Options > Class Explorer**.



Project Manager, Build Configurations and Option Sets

Project Manager

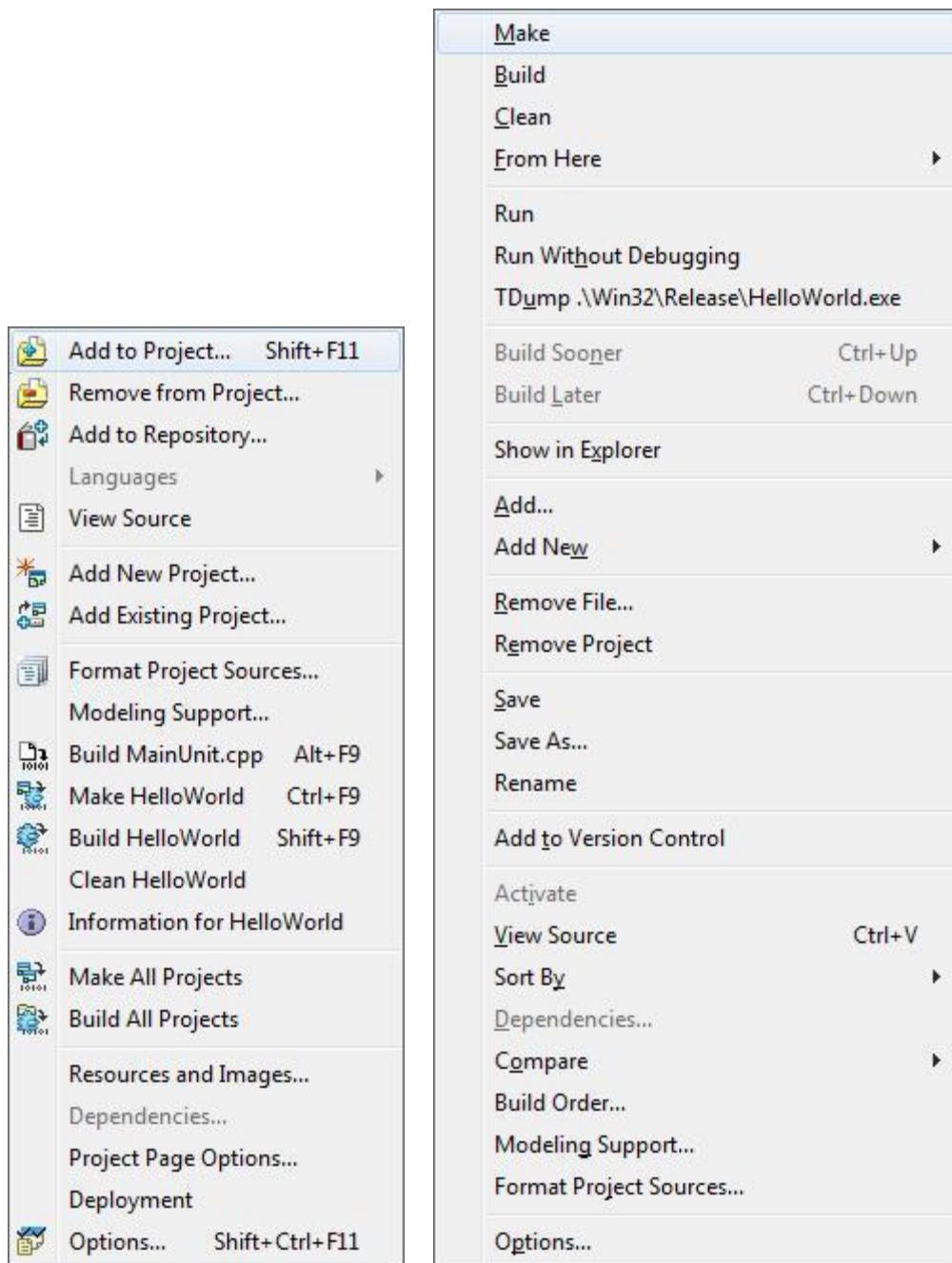
The Project Manager, **View > Project Manager**, displays and organizes the contents of your current project group and any project it contains. You can perform many important project management tasks, such as adding, removing, and compiling files. The default location of the Project Manager is the upper right corner of the IDE, but the window is dockable, as are many windows in the IDE.



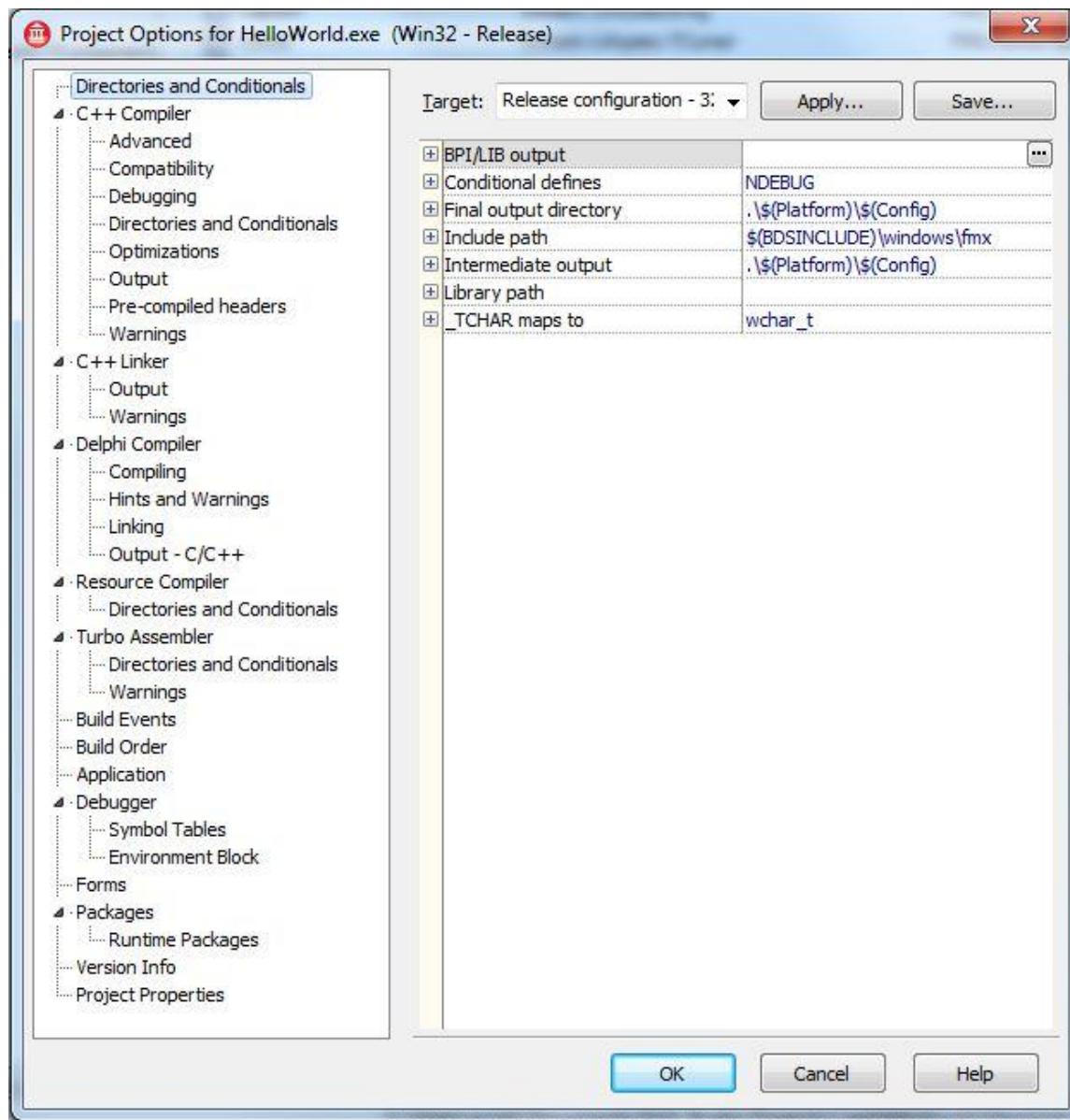
Here is a partial list of the items that can appear in the Project Manager tree structure:

- Project group node
- Project node (typically an .exe file)
- Build Configurations node
- Build configuration (Debug, Release, or custom)
- Target Platforms node (only available for cross-platform applications such as a FireMonkey HD Application or a cross-platform console application): OS X platform (Mac), Windows platform (Win32), Windows platform (Win64)
- Output folder (from Show Project Output context menu command)
- Folder (such as Contains or Requires)
- Source code (.pas or .cpp) that does not contain a form.
- Source code (.pas or .cpp) that contains a form.
- Resource file (.res)
- Form file (.dfm)
- Header file (.h) (C++)
- Import library for a package (.bpi) (C++)

To work with projects, you can use the Project main menu or Right-click a project group in the Project Manager view to display the context menu.



Besides compiling projects, probably the most often used Project menu item is the **Project > Options...** menu item. The pages of the Project Options dialog box enable you to verify and set project-specific options.

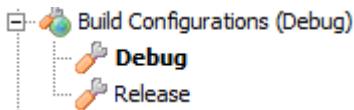


An option value is in boldface if the value differs from the value in its parent's configuration. To revert to the parent configuration value, right-click the option text and click Revert on the context menu. If you change option values, you can save your set of changes in a new configuration or a named option set. You can switch to another configuration or load an option set into the active configuration. Target options, the platform and build configurations and the **Apply** and **Save** buttons are located at the top of the compiler-related options pages.

The Default option is a general option that appears on nearly every page in the Project Options dialog box. Clicking Default saves the current settings on the page as the default for each new project. Note: The Project Options dialog box is resizable.

Build Configurations

Build configurations consist of options that you can set on all the build-related pages of the **Project > Options** dialog box. Build configurations are saved in the project file (such as .dproj or .cbproj). The Project Manager contains a Build Configurations node that lists the available build configurations:

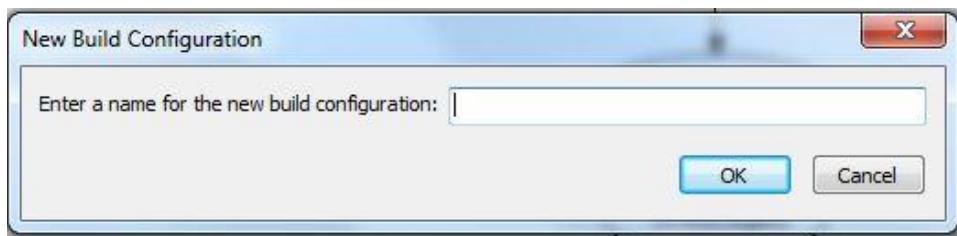


Base, Debug, and Release are the three default build configurations:

- Base acts as a base set of option values that are used in all the configurations you subsequently create. In the Project Manager, the Build Configurations node itself represents the Base configuration.
- The Debug configuration enables optimization and debugging, as well as setting specific syntax options.
- The Release configuration does not produce symbolic debugging information, and the code is not generated for TRACE and ASSERT calls, meaning the size of your executable is reduced.

You can change option values in any configuration, including Base. You can delete the Debug and Release configurations, but you cannot delete the Base configuration. When you compile and save a project, the files are saved in a directory whose name matches the name of the current build configuration. Debug and Release directories exist by default, and a directory is created for any active custom build configuration when you save a project.

You can create new build configurations by Right-Clicking on the Build Configurations node in the Project Manager view and give it a name.



After you have created your new build configuration you can edit the Project options for it by using the **Project > Options...** menu item or Right-Clicking and choosing the "Edit..." context menu choice.

Every project has an active build configuration associated with it, as well as any number of other inactive build configurations that you have created. The name of the active build configuration is displayed in parentheses at the top of the Build Configurations node in the Project Manager, and the active configuration is also displayed in boldface in the list of available configurations in that node. The Compile, Build, and Clean commands use the active build configuration for the project.

To activate a configuration, do either:

- Double-click the configuration in the Project Manager.
- Right-click the configuration and select Activate from the context menu.

Each configuration, except **Base**, is based on another configuration from which it inherits its values. The **Debug** and **Release** configurations inherit their values from **Base**.

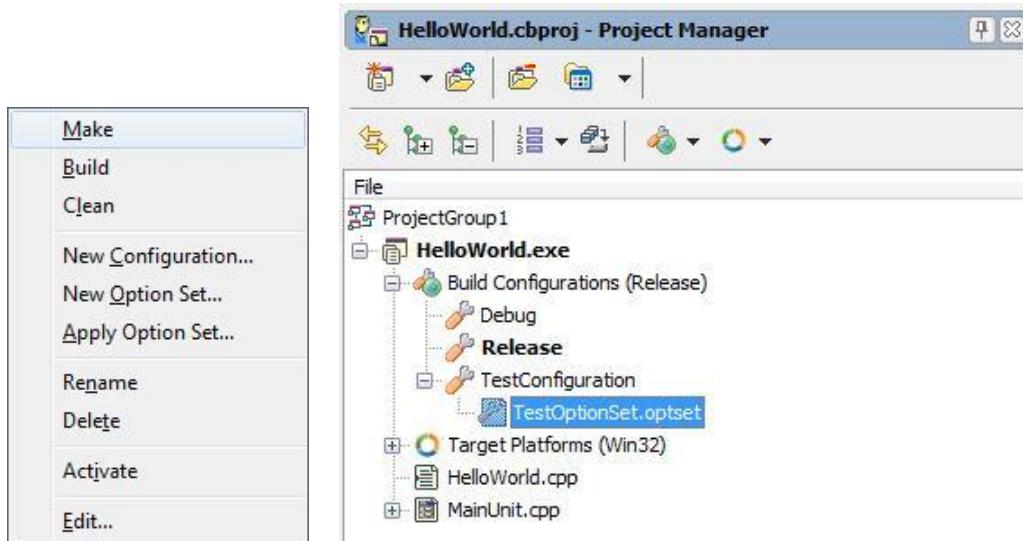
Using the **Configuration Manager**, you can, in turn, create a new configuration based on any given configuration, and the new configuration inherits its option values from its parent. After creating a configuration, you can change its option values to whatever you want, and you can make it the active configuration for a project or projects. You can also delete any configuration except **Base**.

Unless their values are changed, options inherit the values of their parent configuration. This inheritance is not static: if the parent configuration changes, so do inherited values for all its children.

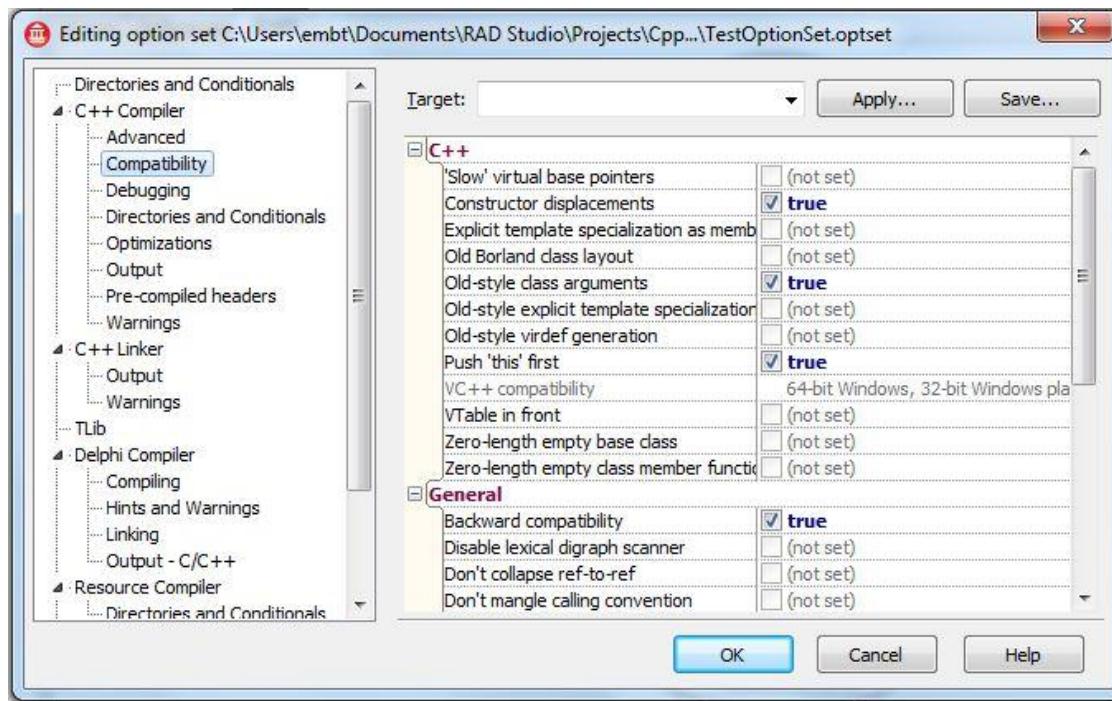
The **default value** of an option is its value in the parent configuration. You can revert an option to its default value by clearing the value in the Project Options dialog box.

Option Sets

You can save a configuration's option values to a file as a **named option set** (an **.optset** file). You can apply an option set to any configuration in any project. You also have the choice to apply an option set by value (applying the option set's values at that one time only) or by reference (so that subsequent changes to the option set are reflected in the configuration) or by value. To create or apply an option set to a build configuration Right-Click on a build configuration node in the Project Window.



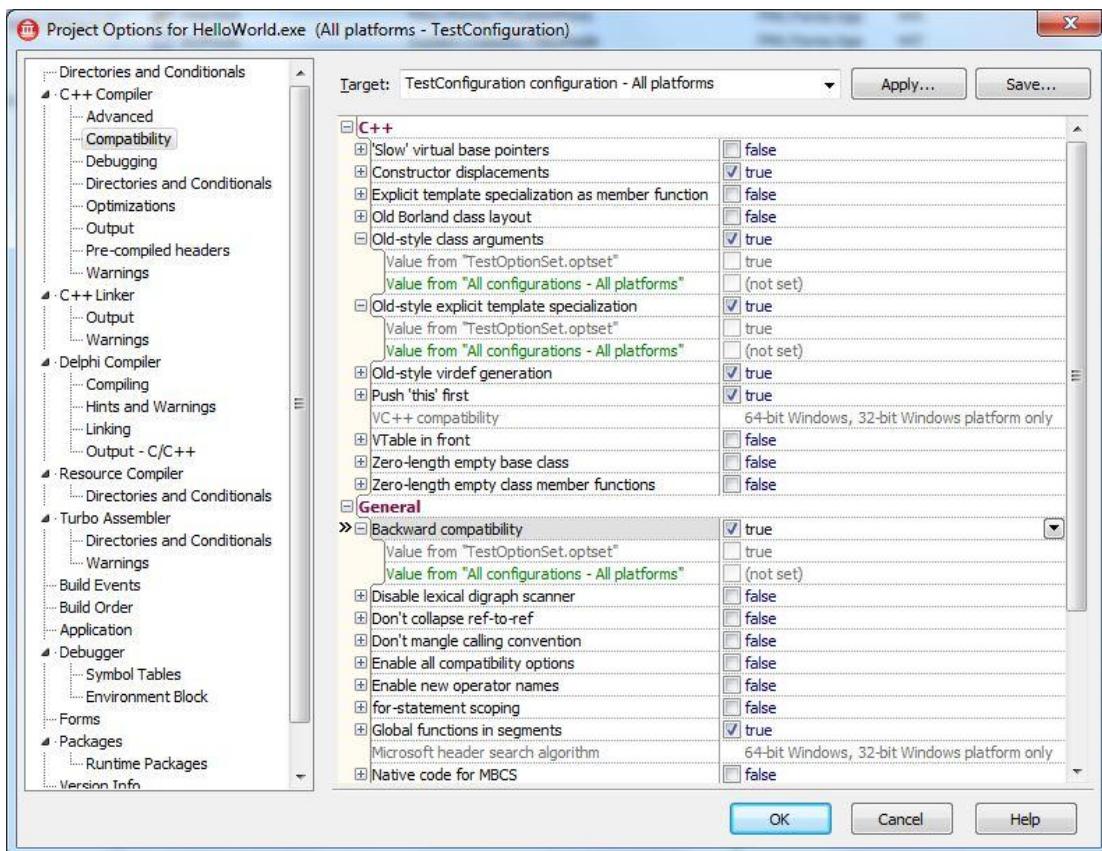
After you create a new option set you can Right-Click on the option set in the Project Window and choose the “Edit...” context menu.



Note that a build configuration is different from an option set, although they are related. Both consist of sets of option values. The main distinction is that configurations are associated with projects, whereas option sets are saved in files independent of projects. Build configuration values are stored in the project file, so saving a project saves changes to its build configurations, but option sets are unaffected. Changing a project's configurations and adding or deleting configurations does not affect option sets. Similarly, saving option sets does not change build configurations.

Each project has its own list of configurations, independent of other projects. However, you can apply any option set to any project. On the **Project Options** dialog box, the **Build Configuration** drop-down list includes all the build configurations for that project - but does not include option sets. The **Project Manager**, on the other hand, lists both configurations and referenced option sets under the Build configurations node.

To see what Project Options are set for a Build Configuration that has an Option Set applied, Right-Click on the build configuration node and choose the “Edit...” context menu item. Then click on any of the plus signs on the right hand pane in the Project Options dialog to see which project option values come from a build configuration and which project option values come from an option set.



Note that configurations and option sets might not contain values for all possible project options - they contain *only* the options that are different from the parent configuration. The **Base** configuration also does not contain values for all possible options.

If an option value is not in a configuration, the IDE looks in its parent configuration, then the parent's parent configuration, and so on. If not found in any of the configurations in the inheritance chain, the value comes from the appropriate tool that is being configured.

For instance, if a configuration inheritance chain does not include a value for a particular compiler option, the default value is specified by the compiler itself. When you save a configuration or option set, only its values are saved - not values for every option.

Compiling and Running the Application

As you develop your application in the IDE, you can compile (or make), build, and run the application in the IDE. All three operations can produce an executable (such as `.exe`, `.dll`, `.obj`, or `.bpl`). However, the four operations differ slightly in behavior:

- **Compile (Project > Compile)** or, for C++, **Make (Project > Make)** compiles only those files that have changed since the last build as well as any files that depend on them. Compiling or making does not execute the application.

E-Learning Series: Getting Started with Windows and Mac Development

- Build (**Project > Build**) compiles all of the source code in the current project, regardless of whether any source code has changed. Building is useful when you are unsure which files have changed, or if you have changed project or compiler options.
- Run (**Run > Run or F9**) compiles any changed source code and, if the compile is successful, executes your application, allowing you to use and test it in the IDE including the use of the Debugger.
- Run without Debugging (**Run > Run Without Debugging** or Shift-Control-F9) compiles the program and executes the application without using the IDE and Debugger.

To delete all of the generated files from a previous build, use the **Clean** command, which is available on the context menu of the project node in the **Project Manager** (**Clean Project** is also available in the Project menu for C++ projects).

To display the current compiler options in the **Messages** window each time you compile your project, choose **Tools > Options > Environment Options** and select **Show command line**. The next time you compile a project, the **Messages** window displays the command used to compile the project and the response file. The response file lists the compiler options and the files to be compiled.

After you compile a project, you can display information about it by choosing **Project > Information**. The resulting **Information** dialog box displays the number of lines of source code compiled, the byte size of your code and data, the stack and file sizes, and the compile status of the project.

As you compile a project, compiler messages are displayed in the **Messages** window. For an explanation of a message, select the message and press **F1**.

When you explicitly build a project, the IDE calls MSBuild, the Microsoft build engine. The build process is entirely transparent to developers. MSBuild is called as part of the Compile, Build, Run and Run Without Debugging commands available on the Project and Run menus. However, you can also invoke MSBuild.exe from the command line or by using the RAD Studio Command Prompt, available on the **Start** menu.

If you enable **Background Compilation** on the Environment Options dialog box, you can run compile and build commands as background threads. See the Embarcadero DocWiki at http://docwiki.embarcadero.com/RADStudio/en/Background_Compilation for additional information about setting the background compilation priority, working while background compilation is taking place and restrictions while background compilation is running.

Debugging the Application

RAD Studio includes a Win32, Win64 and OSX Debugger. The IDE automatically uses the appropriate debugger based on the active project type. Cross-platform debugging within a project group is available using the PAServer and Remote Debugger that are included in RAD Studio as separate installs (see lesson 1 for setup information).

The integrated debuggers let you find and fix both runtime errors and logic errors in your RAD Studio application. Using the debuggers, you can step through code, set breakpoints and watches, and inspect

and modify program values. As you debug your application, the debug windows are available to help you manage the debug session and provide information about the state of your application.

The **Debug Inspector** enables you to examine various data types such as arrays, classes, constants, functions, pointers, scalar variables, and interfaces. To use the **Debug Inspector**, select **Run > Inspect**.

The Debug Desktop is the layout that the IDE uses when you are running your application in Debug mode (F9 in the default key mapping). There is a default Debug Desktop, but you can alternatively select any of the saved desktops to be the Debug Desktop. To set the Debug desktop:

Choose either of the following:

- **View > Desktops > Set Debug Desktop**
-  in the Desktop Toolbar

Stepping Through Code

Stepping through code lets you run your program one line of code at a time. After each step, you can examine the state of the program, view the program output, modify program data values, and continue executing the next line of code. The next line of code does not execute until you tell the debugger to continue.

The **Run** menu provides the **Trace Into** and **Step Over** commands. Both commands tell the debugger to execute the next line of code. However, if the line contains a function call, **Trace Into** executes the function and stops at the first line of code *inside* the function. **Step Over** executes the function and then stops at the first line *after* the function.

The main toolbar also contains buttons for debugging operations (from left to right):



- Run Without Debugging (Shift-Ctrl-F9)
- Run With Debugging (F9)
- Pause the application
- Program Reset (Ctrl-F2)
- Trace Into (F7)
- Step Over (F8)
- Run Until Return (Shift-F8)

Evaluate Modify

The Evaluate/Modify functionality allows you to evaluate an expression. You can also modify a value for a variable and insert that value into the variable. The Evaluate/Modify functionality is customized for the language you are using:

- For a C++ project, the Evaluate/Modify dialog accepts only C++ expressions.

- For a Delphi project, the Evaluate/Modify dialog accepts only Delphi expressions.

Breakpoints

Breakpoints pause program execution at a certain point in the program or when a particular condition occurs. You can then use the debugger to view the state of your program, or step over or trace into your code one line or machine instruction at a time. The debugger supports four types of breakpoints:

- Source breakpoints pause execution at a specified location in your source code.
- Address breakpoints pause execution at a specified memory address.
- Data breakpoints allow you to pause execution when the contents changes for memory at a particular address. Data breakpoints are available only for the Win32 debugger. Data breakpoints are automatically disabled when a debugging session ends, because the address of a variable can change from one debug session to the next. To re-use a data breakpoint during a subsequent debugging session, you need to re-enable the data breakpoint after your debugging session begins.
- Module load breakpoints pause execution when the specified module loads.

During a debugging session, any line of code that is eligible for a breakpoint is marked with a blue dot  in the left gutter of the Code Editor.

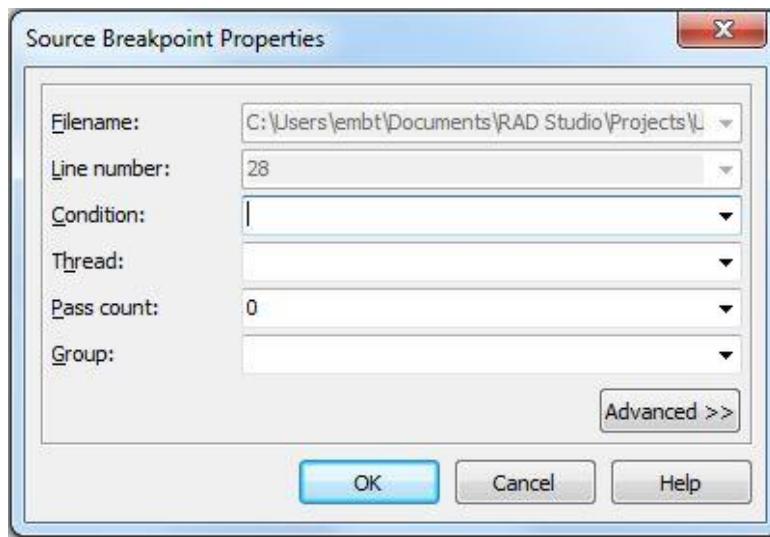
You can also set breakpoints on frames displayed in the Call Stack window. The breakpoint icons in the Call Stack window are similar to those in the Code Editor, except that the blue dot  in the Call Stack indicates only that debug information is available for the frame, not whether a breakpoint can be set on that frame.

Breakpoints are displayed in the Breakpoint List window, available by selecting **View > Debug windows > Breakpoints**.

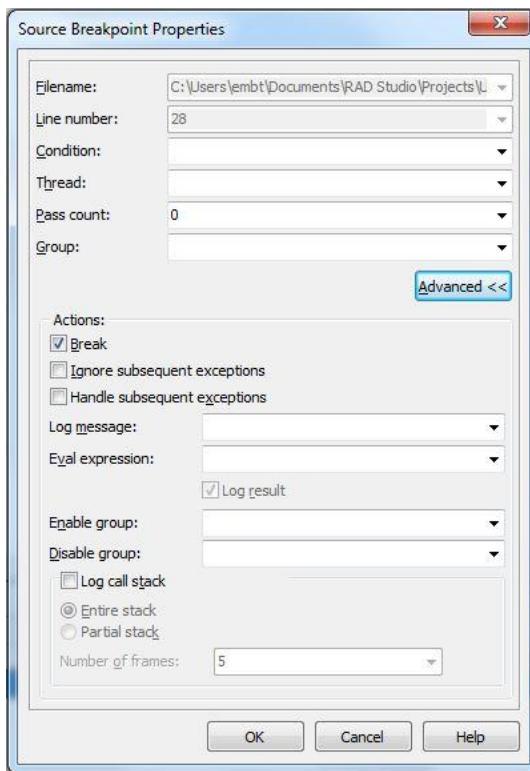
You can drag breakpoint icons and drop them in the Code Editor window; a moved breakpoint retains the settings of the original breakpoint. The following icons are used to represent breakpoints in the Code Editor gutter.

To set a source breakpoint

- To pre-fill the line number in the dialog box, click the line of source in the **Code Editor** at the point where you want to stop execution.
- Choose **Run > Add Breakpoint > Source Breakpoint** to display the **Add Source Breakpoint** dialog box. **Tip:** To change the **Code Editor** gutter, choose **Tools > Options > Editor Options > Display** and adjust the **Gutter width** option.
- In the **Add Source Breakpoint** dialog box, the file name is prefilled with the name of the file, and **Pass count** is set to 0 (meaning that the breakpoint fires on the first pass). In the **Line number** field, enter the line in the **Code Editor** where you want to set the breakpoint.



- To apply a condition to the address breakpoint, enter a conditional expression in the **Condition** field. The conditional expression is evaluated each time the breakpoint is encountered, and program execution stops when the expression evaluates to True.
- To associate the source breakpoint with a breakpoint group, enter the name of a group or select from the **Group** drop-down list.
- To set any of the **Advanced** options click the “Advanced >>” button in breakpoint properties dialog.

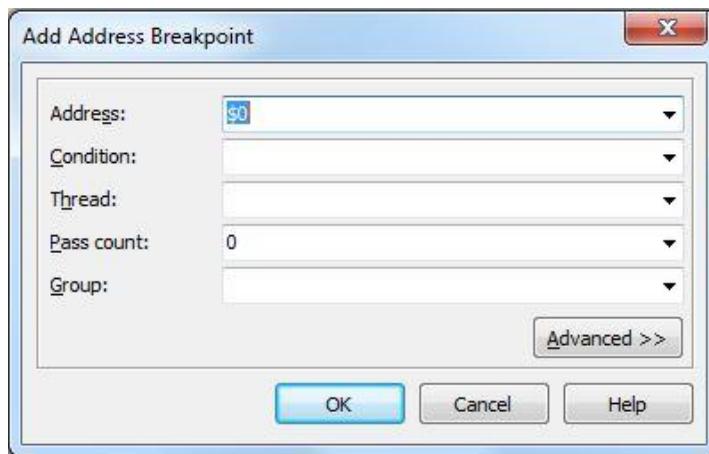


Advanced source breakpoints allow you to Break, Ignore/Handle exceptions, Log Messages, Evaluate an Expression (including call a function with side effects, set a variable), and Enable/Disable groups of breakpoints.

Note: To quickly set a breakpoint on a line of source code, click the left gutter of the **Code Editor** next to the line of code where you want to pause execution.

To set an address breakpoint

- Run your application in Debug mode (for example, use F9, F8, F7, or F4).
- Choose **Run > Add Breakpoint > Address Breakpoint** to display the **Add Address Breakpoint** dialog box.
- In the **Address** field, enter the address in memory (such as \$00011111) at which you want to set the breakpoint.
- To apply a condition to the address breakpoint, enter a conditional expression in the **Condition** field. The conditional expression is evaluated each time the breakpoint is encountered, and program execution stops when the expression evaluates to true.
- To specify that the address breakpoint will only fire after a number of passes, enter the number in the **Pass count** field.
- To associate the address breakpoint with an existing breakpoint group, enter the group name in the **Group** field, or select the name of an existing group from the drop-down list.
- To set any of the **Advanced** options, see the Embarcadero DocWiki's "Add Address Breakpoint or Add Data Breakpoint" section at http://docwiki.embarcadero.com/RADStudio/en/Add_Address_Breakpoint_or_Add_Data_Breakpoint.

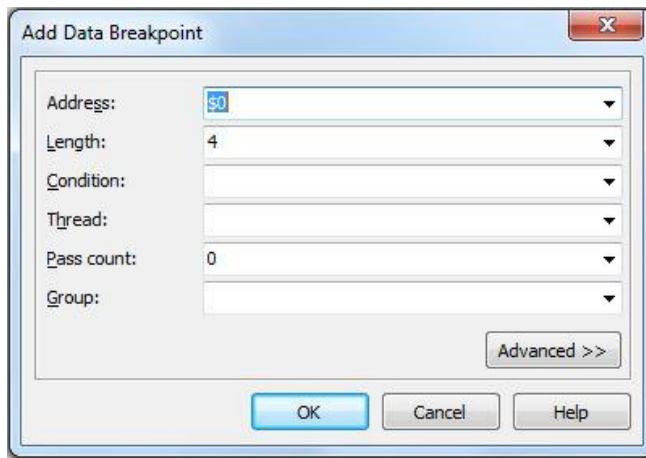


Note: You can also set an address breakpoint in the **CPU view** or the **Disassembly view** by clicking in the gutter.

To set a data breakpoint

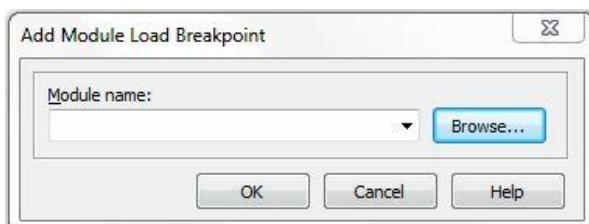
- Run your application in Debug mode (for example, use F9, F8, F7, or F4).

- Choose **Run > Add Breakpoint > Data Breakpoint** to display the **Add Data Breakpoint** dialog box.
- In the **Address** field, enter the address of the data you want to function as the data breakpoint.
- In the **Length** field, specify the length of the data operand that is to constitute the breakpoint. Note that a warning is displayed for the following issues:
 - The length of the data breakpoint should not cross an even-byte boundary. (A data breakpoint with a 1-byte length has no alignment problems, but 2-byte and 4-byte data breakpoints might cover more or fewer addresses than you intend.)
 - The data breakpoint should not be set on a stack location. (The breakpoint might be hit so often that the program cannot run properly.)
- To apply a condition to the breakpoint, enter a conditional expression in the **Condition** field. The conditional expression is evaluated each time the breakpoint is encountered, and program execution stops when the expression evaluates to True.
- To specify that the breakpoint only fires after a number of passes, enter the number in the **Pass count** field.
- To associate the data breakpoint with an existing breakpoint group, enter the group name in the **Group** field, or select the name of an existing group from the drop-down list.
- To set any of the **Advanced** options, see the Embarcadero DocWiki's "Add Address Breakpoint or Add Data Breakpoint" section at http://docwiki.embarcadero.com/RADStudio/en/Add_Address_Breakpoint_or_Add_Data_Breakpoint.



To set a module load breakpoint

Choose **Run > Add Breakpoint > Module Load Breakpoint** to display the **Add Module Load Breakpoint** dialog box.



Do either of the following:

- In the **Module name** field, enter the name of the DLL, package, or other module type that you want to monitor, or select a name from the drop-down list, and click **OK**.
- Click **Browse** to open the **Select Module** dialog box and browse modules from another target platform. Click the **Modules** dropdown next to the **File Name** field, and select the type of modules you want to see:
 - Modules (*.dll, *.ocx, *.bpl, *.exe, *.dylib) {both Windows and Mac modules}
 - Windows Modules (*.dll, *.ocx, *.bpl, *.exe) {only Windows modules}
 - Mac OS X Modules (*.dylib) {only Mac modules}
 - Any file *.*

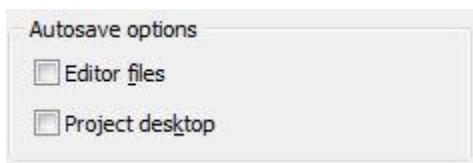
Click **Open** to open the module you want. The name is automatically inserted in the Add or Edit Module Load Breakpoint dialog box.

Note: You can also use the **Modules Window** to set a module load breakpoint.

When the module you specify is loaded during program execution, the module load breakpoint is hit, and program execution pauses.

To persist breakpoints from session to session

If you set the **Autosave Project desktop** option, breakpoints you set for a project will persist from session to session.



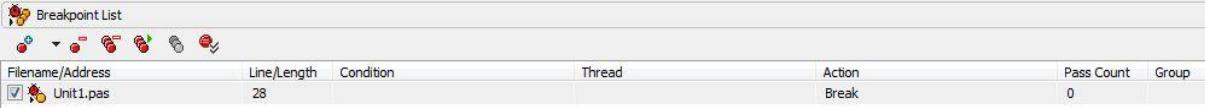
1. Set the breakpoints (and watches) that you want to keep from session to session.
2. Select **Tools > Options > Environment Options**.
3. Enable **Autosave Project desktop**. When you exit the product or close your project, your desktop settings are saved to a .dsk file. When you reopen your project, the product reads the .dsk file and restores your saved desktop, breakpoints, watches and open files.

You need to delete the .dsk file when you no longer want the saved set of breakpoints to persist in your project (or any other items controlled by the **Autosave Project desktop** option).

To modify a breakpoint

1. Open the **Breakpoint List Window** by selecting **View > Debug Windows > Breakpoints**. Right-click the icon for the breakpoint you want to modify. For a source breakpoint, you can right-click the breakpoint icon in the **Code Editor** gutter, and choose **Breakpoint Properties**.

2. Set the options in the **Breakpoint Properties** dialog box to modify the breakpoint. For example, you can set a condition, create a breakpoint group, or specify an action that is to occur when execution reaches the breakpoint.
3. Click **Help** for more information about the options in the dialog box.
4. Click **OK**.



Breakpoint List						
Filename/Address	Line/Length	Condition	Thread	Action	Pass Count	Group
Unit1.pas	28			Break	0	

To create a breakpoint group

1. Open the Breakpoints List by choosing View > Debug Windows > Breakpoints.
2. Right-click the breakpoint and choose Breakpoint Properties.
3. To create a breakpoint group, enter a group name in the Group field. To add the breakpoint to an existing group, select a name from the drop-down list box.
4. Click **OK**.

To enable or disable a breakpoint or a breakpoint group

1. Right-click the breakpoint icon in either the **Code Editor** or the **Breakpoint List Window** and choose Enabled to toggle between enabled and disabled. In the **Breakpoint List**, you can click the checkbox at the left of the icon.
2. To enable or disable all breakpoints, right-click a blank area (not on a breakpoint) in the **Breakpoint List** window and choose **Enable All** or **Disable All**.
3. To enable or disable a breakpoint group, right-click a blank area (not on a breakpoint) in the **Breakpoint List** window and choose **Enable Group** or **Disable Group**.

Tip: Press the Ctrl key while clicking a breakpoint in the **Code Editor** gutter to toggle between enabled and disabled. Disabling a breakpoint or breakpoint group prevents it from pausing execution, but retains the breakpoint settings, so that you can enable it later.

To create a conditional breakpoint

1. Choose **Run > Add Breakpoint** and select the type of breakpoint you want from the submenu.
2. Complete the fields in the dialog box as described in the procedure given earlier for that breakpoint type.
3. In the **Condition** field, enter a conditional expression to be evaluated each time this breakpoint is encountered during program execution. The breakpoint pauses execution when the expression evaluates to True.
4. Complete other fields as appropriate.
5. Click **OK**.

Conditional breakpoints are useful when you want to see how your program behaves when a variable falls into a certain range or what happens when a particular flag is set.

If the conditional expression evaluates to True (or not zero), the debugger pauses the program at the breakpoint location. If the expression evaluates to False (or zero), the debugger does not stop at the breakpoint location.

To set a breakpoint on a specific thread

1. Choose **Run > Add Breakpoint** and select the type of breakpoint you want from the submenu.
2. Complete the fields in the dialog box as described in the procedure given earlier for that breakpoint type.
3. In the **Thread** field, enter or select the thread number (for numbered threads) or thread name (for named threads).
4. Complete other fields as appropriate.
5. Click **OK**.

To associate actions with a breakpoint

1. On the **Breakpoint List Window**, right-click the breakpoint and choose **Breakpoint Properties**.
2. Click **Advanced** to display additional options.
3. Check the actions that you want to occur when the breakpoint is encountered. For example, you can specify an expression to be evaluated and write the result of the evaluation to the **Event Log**.
4. Click **OK**.

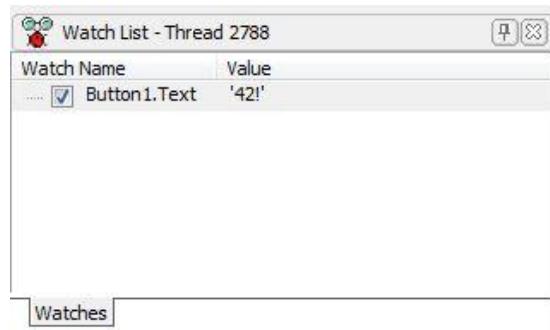
To change the color of the text at the execution point or the color of breakpoints

1. Choose **Tools > Options > Editor Options > Color**.
2. In the code sample window, select the appropriate language tab. For example, to change the breakpoint color for Delphi code, select the Delphi tab.
3. Scroll the code sample window to display the execution and breakpoint icons in the left gutter of the window.
4. Click anywhere on the execution point or breakpoint line that you want to change.
5. Use the Foreground Color and Background Color drop-down lists to change the colors associated with the selected execution point or breakpoint.
6. Click **OK**.

Note: You can also set breakpoints in the **Breakpoint List**, the **CPU** window (and the **Disassembly** view), the **Call Stack** view, and the **Modules** window.

Watches

Watches let you track the values of program variables or expressions as you step over or trace into your code. As you step through your program, the value of the watch expression changes if your program updates any of the variables contained in the watch expression.



To add variables or expressions to the Watch Window, select the variable or expression and hit **Ctrl-F5** or Right-Mouse-Click and choose **Debug > Add Watch at Cursor** from the context menu.

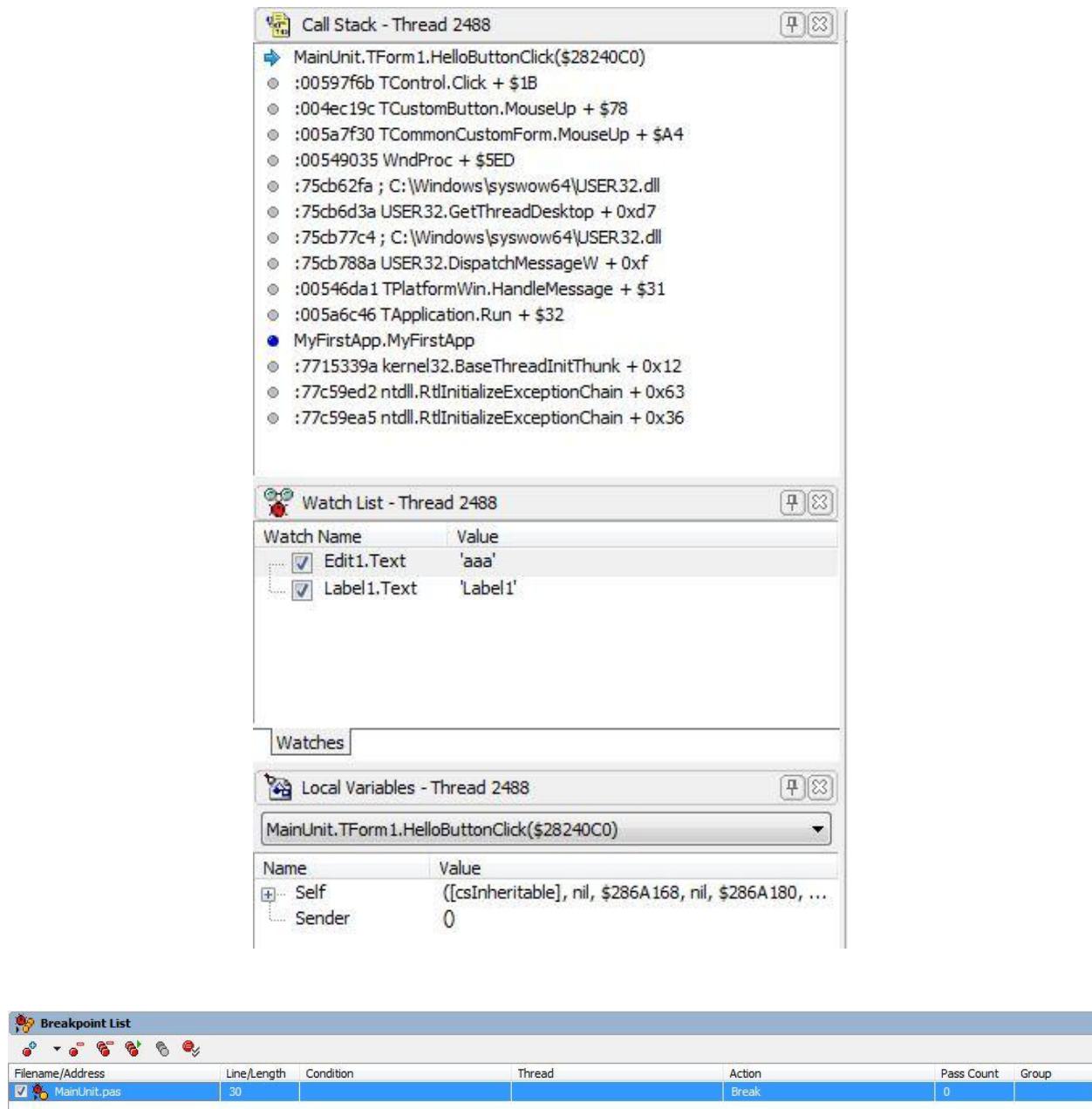
Debug Windows

The following debug windows are available to help you debug your program. By default, most of the windows are displayed automatically when you start a debugging session. You can also view the windows individually by selecting **View > Debug Windows**.

Each window provides one or more right-click context menus. The **F1** Help for each window provides detailed information about the window and the context menus.

- Breakpoint List - Displays all of the breakpoints currently set in the Code Editor or CPU window.
- Call Stack - Displays the current sequence of function calls.
- Watch List - Displays the current value of watch expressions based on the scope of the execution point.
- Local Variables - Displays the current function's local variables, enabling you to monitor how your program updates the values of variables as the program runs.
- Modules - Displays processes under control of the debugger and the modules currently loaded by each process. It also provides a hierarchical view of the namespaces, classes, and methods used in the application.
- Threads - Displays the status of all processes and threads of execution that are executing in each application being debugged. This is helpful when debugging multi-threaded applications. For Windows Vista, the Threads Status includes a Wait Chain column that lists thread blockages and deadlocks.
- Event Log - Displays messages that pertain to process control, breakpoints, output, threads, and module.
- CPU Windows - Displays the low-level state of your program, including the assembly instructions for each line of source code and the contents of certain registers.
- FPU - Displays the contents of the Floating-point Unit and SSE registers in the CPU.

E-Learning Series: Getting Started with Windows and Mac Development



CPU / Thread #2488

```

005AA70A 55 push ebp
005AA70B 6873A75A00 push $005aa773
005AA710 64FF30 push dword ptr fs:[eax]
005AA713 648920 mov fs:[eax],esp
MainUnit.pas.30: Label1.Text := 'Hello ' + Edit1.Text + ' !';
005AA716 688CA75A00 push $005aa78c
005AA718 8D55F0 lea edx,[ebp-$10]
005AA71E 8B45FC mov eax,[ebp-$04]
005AA721 8B80B4010000 mov eax,[eax+$000001b4]
005AA727 8B08 mov ecx,[eax]
005AA729 FF91DC010000 call dword ptr [ecx+$000001dc]
005AA72F FF75F0 push dword ptr [ebp-$10]
005AA732 68A8A75A00 push $005aa7a8
005AA737 8D45F4 lea eax,[ebp-$0c]
005AA73A BA03000000 mov edx,$00000003
005AA73F E8D8D4E5FF call @UstrcmpN
005AA744 8B55F4 mov edx,[ebp-$0c]
005AA747 8B45FC mov eax,[ebp-$04]
005AA74A 8B80B8010000 mov eax,[eax+$000001b8]
005AA750 8B08 mov ecx,[eax]
005AA752 FF91D8010000 call dword ptr [ecx+$000001d8]
MainUnit.pas.31: end;
005AA758 33C0 xor eax,eax
005AA75A 5A pop edx
005AA75B 59 pop ecx
005AA75C 59 pop ecx
005AA75D 648910 mov fs:[eax],edx
005AA760 687AA75A00 push $005aa77a
005AA765 8D45F0 lea eax,[ebp-$10]
005AA768 BA02000000 mov edx,$00000002
005AA76D F84A75E5FF call @UstrcmpN

00410000 50 DC 5C 00 7C FF 00 00 PÜ\.jÿ..
00410008 50 DC 5C 00 7D FF 00 00 PÜ\.fÿ..
00410010 50 DC 5C 00 7E FF 00 00 PÜ\.~ÿ..
00410018 50 DC 5C 00 7F FF 00 00 PÜ\.~ÿ..
00410020 50 DC 5C 00 60 FF 00 00 PÜ\.ÿ..
00410028 50 DC 5C 00 61 FF 00 00 PÜ\.ay..
00410030 50 DC 5C 00 62 FF 00 00 PÜ\.by..
00410038 50 DC 5C 00 63 FF 00 00 PÜ\.cy..
00410040 50 DC 5C 00 64 FF 00 00 PÜ\.dy..
00410048 50 DC 5C 00 65 FF 00 00 PÜ\.ey..
00410050 50 DC 5C 00 66 FF 00 00 PÜ\.fy..

```

EAX	00000000	CF 0
EBX	028240C0	PF 1
ECX	00000000	AF 0
EDX	028240C0	ZF 1
ESI	028240C0	SF 0
EDI	004E5158	TF 0
EBP	0018FC68	IF 1
ESP	0018FC4C	DF 0
EIP	005AA716	OF 0
EFL	00200246	IO 0
CS	0023	NF 0
DS	002B	RF 0
0018FC00	0057D9EC	.WÜi
0018FCBC	02836950	.fiP
0018FCB8	000205F0	...ð
0018FCB4	42440000	BD..
0018FCB0	41200000	A ..
0018FCAC	004EC19C	.NÄæ
0018FCA8	0018FCC8	..üÈ
0018FCA4	00000000
0018FCA0	41200000	A ..
0018FC9C	42440000	BD..
0018FC98	41B00000	A°..
0018FC94	42D20000	BÒ..
0018FC90	00000000
0018FC8C	00000000
0018FC88	00000000
0018FC84	028240C0	.,@A
0018FC80	005984E6	Y,æ
0018FC7C	00000000
0018FC78	028240C0	.,@A
0018FC74	004EBF47	.N_G
0018FC70	00000000
0018FC6C	00597F68	Y,k
0018FC68	0018FCA8	..ü"
0018FC64	02836950	.fiP
0018FC60	028240C0	.,@A
0018FC5C	00000000
0018FC58	00000000
0018FC54	0018FC68	..üh
0018FC50	005AA773	.Z§s
0018FC4C	0018FCDB	..üØ
0018FC48	77CC0F3B	wí.;
0018FC44	00020045	..E

FPU - Thread 2488

IPT: \$58671E OPCODE: \$D85A OPTR: \$18FC98		IM 0	IE 0
ST0	Empty	DM 1	DE 0
ST1	Empty	ZM 0	ZE 0
ST2	Empty	OM 0	OE 0
ST3	Empty	UM 1	UE 0
ST4	Empty	PM 1	PE 1
ST5	Empty	PC 3	SF 0
ST6	Empty	RC 0	ES 0
ST7	Empty	IC 1	C0 1
CTRL	1372		C1 0
XMM0	00000000000000000000000000000000		C2 0
XMM1	00000000000000000000000000000000		ST 0
XMM2	00000000000000000000000000000000		C3 0
XMM3	00000000000000000000000000000000		BF 0
XMM4	00000000000000000000000000000000		
XMM5	00000000000000000000000000000000		
XMM6	00000000000000000000000000000000		
XMM7	00000000000000000000000000000000		
MXCSR	00001F80		

Summary, Looking Forward, To Do Items, Resources, Q&A and the Quiz

In Lesson 3 you learned how to leverage the features of the Integrated Development Environment (IDE) to help you build successful applications. The IDE has an extensible architecture allowing you to customize your working desktops, control the writing of code, explore the structure of your applications, rapidly create the UI of your program, add third party tools and create and install components.

Lesson 4 will cover an introduction to the Delphi and C++ programming languages. These modern programming languages have a wide range of capabilities that are well documented in numerous books, videos and online tutorials. Lesson 4 will focus on the language features that support RAD, visual and component based development.

Until the next lesson, here are some things you can try, articles to read and videos to watch to enhance what you learned in this lesson and to prepare you for the next lesson.

To Do Items

Explore the rest of the IDE that wasn't covered in Lesson 3. Take a look at each of the categories and items in the Object Repository and familiarize yourself with project and file types. Customize your **File > New** menu to have the items that you will most often use in your daily work. Create your own desktop layouts and save them for later use. Load some of the sample projects that are included with RAD Studio and explore the **View** menu when a project is loaded. Set breakpoints in the sample code and run the applications with debugging to explore the debugger capabilities (unless you write absolutely 100% perfect code all of the time, you'll live part of your life in the debugger).

Links to Additional Resources

- The course landing page URL is <http://www.embarcadero.com/firemonkey/firemonkey-e-learning-series>
- Download and Watch Alistair Christie's CodeRage III video – 100 Delphi IDE Tips and Hints: <http://cc.codegear.com/Download.aspx?id=26411>
- Read about new IDE capabilities that have appeared in the XE2, XE and 2010 releases: http://docwiki.embarcadero.com/RADStudio/en/What's_New_in_Delphi_and_C%2B%2BBuilder_2010, http://docwiki.embarcadero.com/RADStudio/en/IDE_Changes_for_XE, http://docwiki.embarcadero.com/RADStudio/en/IDE_Changes_for_XE2
- Watch the debug visualizers video - <http://www.youtube.com/watch?v=qGE8WQQoKho>

Q&A:

Here are some of the answers for the questions I've received (so far) for this lesson. I will continue to update this Course Book during and after course. (To be added after the lesson webinars take place).

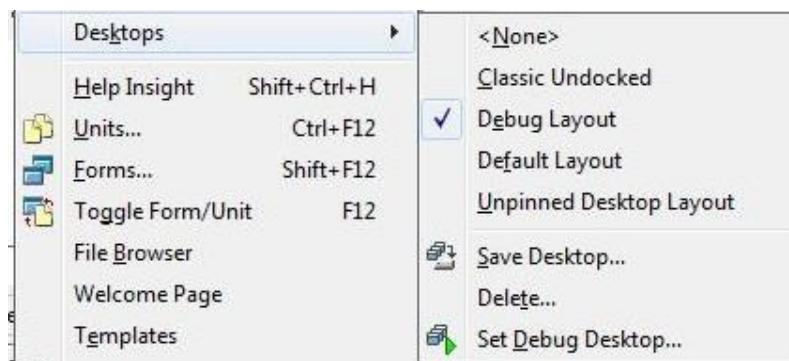
E-Learning Series: Getting Started with Windows and Mac Development

Q: VCL versus FireMonkey application, which one is better in terms of less memory usage?

A: VCL and FireMonkey applications grow and shrink based on which units you link into your application executable. If you want to have smaller executables you can choose to build with or without runtime packages.

Q: You showed how to create and save new desktop layout? But is there a way to delete one you have created and don't want to keep. I have spent a big deal of time trying to figure that out by myself.

A: Use the **View > Desktops > Delete...** menu item to remove a saved desktop layout.



Q: When folding source code, in earlier versions, a print still printed on printer the unfolded code. Is it now possible to have a print of folded code?

A: Printing the source file will print the file contents. If you want to print the folded code, capture the code editor with a screen capture tool.

Q: I have XE2 Professional installed but I don't have the Xcode under tools is not shown.

A: Are you referring to 'dpr2xcode' in the Tools menu. David added that himself, by customizing the Tools menu – this is covered in Lesson 1.

Q: Regarding the question on how to print debug info: You can use Code Insight as an awesome logging tool. It supports saving everything

A: Yes, Code Insight Express Edition from Raize Software is included in XE2.

Q: Sometimes we use ExtractFilePath and Application.ExeName to navigate through external resources. .ExeName property is not available in FireMonkey. What do we use in order to get current directory?

A: The System.IOUtils unit contains classes and methods that provide access to directory, path, attributes and other folder and file information. You can find more information about IOUtils on the Embarcadero DocWiki at <http://docwiki.embarcadero.com/Libraries/en/System.IOUtils>.

Q: I like the fact that you can move the current position by dragging the little arrow, while the app is running.

A: Yes, you can change the execution point in your code by dragging the execution arrow on a breakpoint to another source line. But, be careful that you understand any side effects that might take place because of you changing the execution point.

Q: When David uses the term FireMonkey HD, what does the HD stand for?

A: High Definition. Just like with HDTV, HD uses for iPad as opposed to iPhone, etc.

E-Learning Series: Getting Started with Windows and Mac Development

Q: Custom layouts work great with more than one monitor!

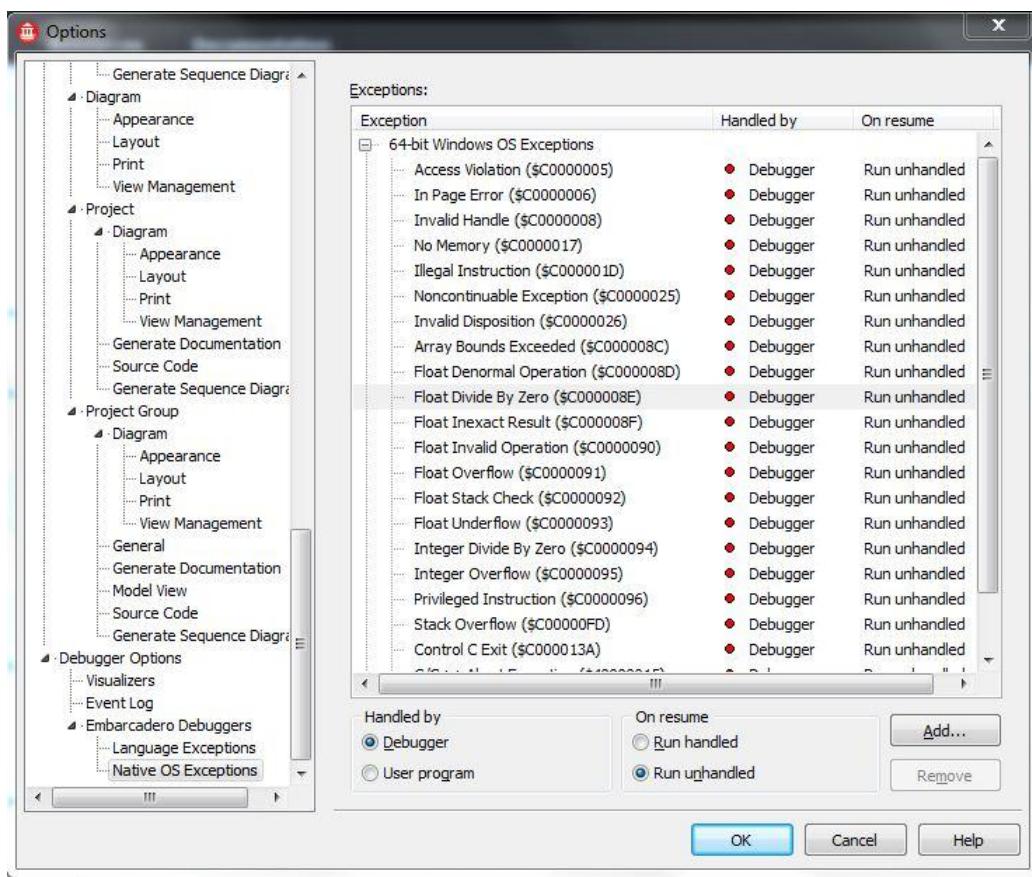
A: Thanks! Good feedback!

Q: I think this question comes up a lot. Can you verify? I have Xcode 4.2.1 installed and working with FireMonkey and Free Pascal. All is well. Can I install the latest version of Xcode without losing any FireMonkey capabilities? Are there any tips/tricks I should watch for or do? Thank you so much!

A: If you don't *need* to install Xcode 4.3.x, don't. It'll save you some troubles for now. If you have to install it, make sure you do NOT uninstall 4.2.x when the 4.3.x installer asks - because it will remove the entire /Developer directory without asking - nuking FPC/FMI in the process causing you a lot of re-installation pain.

Q: Breakpoints are fine... but what about exceptions, e.g. divide by 0. What do I have to set so that I'm pointed to the failing statement?

A: You can choose to have the debugger break on exceptions using the **Tools > Options > Debugger Options** settings for **Embarcadero Debuggers > Language Exceptions** and **Embarcadero Debuggers > Native OS Exceptions**.



Additional information is available on the Embarcadero DocWiki at

http://docwiki.embarcadero.com/RADStudio/en/Debugger_Exception_Notification.

E-Learning Series: Getting Started with Windows and Mac Development

Q: I was trying to drag from toolbar a TLabel component on a TTabItem control within a TTabControl. However, the TLabel did not stay with the specific TTabItem control. How do you get this association to work?

A: I added a TTabControl onto my form. I added three TTabItem(s) to the TTabControl using the component editor (Right-Mouse-Click on the TTabControl in the form designer). I added TLabel components into a TTabControl and the TLabel(s) were placed in the currently selected TTabItem – which was the first TTabItem. You use the Structure view to move components to other TTabItems.

Q: You did something interesting when you placed a FireMonkey TLabel inside a FireMonkey edit box, then moved the TLabel outside the edit box to place it relative to the edit box. The edit box is apparently not a real container like a VCL TPanel. What would you do if you wanted a label to stay outside of a Panel, and move with the panel?

A: I tried compositing a TLabel with a TPanel using FireMonkey and it worked. First, I added a TPanel to my FireMonkey HD application. Next, I added a TLabel to the form and in the Structure view I dragged the TLabel into the TPanel. Then, in the form designer, I dragged the TLabel outside the TPanel. Then, when I dragged the TPanel to a different location in the form designer, the TLabel followed with the TPanel. With FireMonkey you can choose to composite a component inside another component or place it outside of the component and it still is parented to that component.

Q: What happens if you use a Windows statement that is not supported on OSX. One example may be Application.ProcessMessages used to update the display prior to a long process.

A: The TApplication class is included in the runtime library. You can call ProcessMessages and use other TApplication class methods in your FireMonkey applications for Windows and Mac.

Q: Is there any way to keep the non-visual components from cluttering up the form?

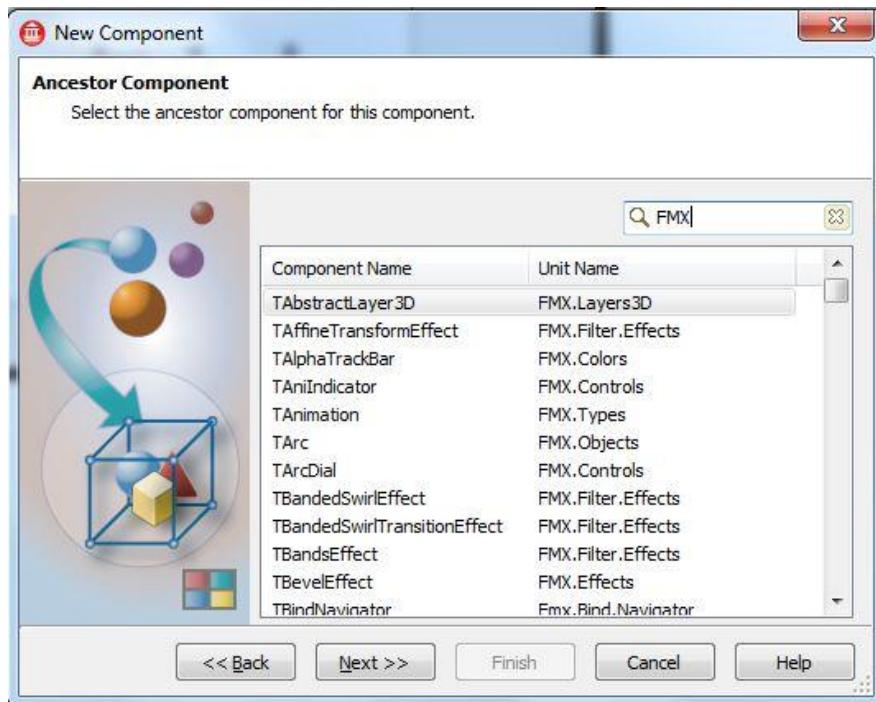
A: There isn't any place to put non-visual components on a form that is out of the way – other than at the extreme borders. You can choose to put non-visual components, like Data Access components, in a Data Module.

Q: How are deployment package files built?

A: You can create design time and run time packages with Delphi and C++. Use the **File > New > Other...** menu item and choose "Package" from the "C++Builder Projects" or "Delphi Projects" categories. Additional information is available on the Embarcadero DocWiki at http://docwiki.embarcadero.com/RADStudio/en/Working_with_Packages_and_Components_-_Overview.

Q: How do you create a custom control with the possibility to edit its style later?

A: You can create FireMonkey components using the **Component > New Component** menu to create a new component from any installed FireMonkey components as well as the base **TFMXObject** class. Once your new component is implemented and installed, you can use it in your applications and have full access to changing the style as needed.



You can read more about creating FireMonkey components on the Embarcadero DocWiki at http://docwiki.embarcadero.com/RADStudio/XE2/en/FireMonkey_Components_Guide.

Q: Why would I continue to use frames rather than composite components?

A: With FireMonkey you can composite components inside other components. So, there is no need for frames containing components. You can create super-components with FireMonkey and reuse them in your applications.

Q: Is there a visual cue that shows that a derived form has been changed? This would be REALLY helpful.

A: No there isn't a visual cue that properties are changed from their inherited values. You can revert a changed property back to its inherited value by selecting the component on the inherited form and choose "Revert from Inherited" in the context menu.

Q: When a project is saved to repository, is it transferred when we upgrade to the next compiler? E.g., projects stored to the XE repository are migrated to XE2?

A: You will need to copy over the XML files for the template projects from your previous version's repository folder.

Q: Will FireMonkey be any help in Web Application Development?

A: FireMonkey is used to create native code applications that run on Windows, Mac and iOS using the CPU and GPU. FireMonkey is not used for building browser based applications or server side applications. Your client applications can use Internet protocols to connect to Web services and DataSnap server multi-tier applications. You can leverage FireMonkey and the runtime library non-visual components to do server side processing for example with image processing.

E-Learning Series: Getting Started with Windows and Mac Development

Q: Why is this webinar more focused on getting started with C++ Builder and not the “in-depth” and “what’s new” in FireMonkey?

A: Because it's targeted to new developers and people using the trial. We have many other webinars and videos for the “what’s new in FireMonkey” and advanced topics like FireMonkey 3D and iOS development. The lessons will also get more advanced later in the series. This is lesson 3 of 9 lessons (with more being planned for the future – send me any ideas you have for advanced topics you'd like to see).

Q: Will this webinar have more 3D FireMonkey applications and examples?

A: FireMonkey 3D will be covered in Lesson 7.

Q: What email account to use for the extension of the FireMonkey trial period?

A: Send an email to davidi@embarcadero.com

Q: When will FireMonkey support iOS?

A: FireMonkey for iOS is already supported for Delphi using the Windows IDE, Free Pascal compiler for ARM processor and Xcode (See lesson 1 for setup and configuration). John Thomas, Director of Product Management for Developer Tools recently wrote an EDN article titled “Coming soon to a RAD IDE near you, the future of C++ - 64bit, C++11, ARM, iOS and Android” that outlines our plans for C++ support for Win64, iOS and Android. You'll find the article on EDN at <http://edn.embarcadero.com/article/42275>.

If you have any additional questions – send me an email - davidi@embarcadero.com

Self Check Quiz

1) Which view is not included in RAD Studio's View Menu?

- a) Project Manager
- b) Desktops
- c) Welcome Page
- d) Structure
- e) Rooms
- f) Object Inspector

2) Which desktop layout is not one of the standard built-in layouts?

- a) <none>
- b) Default Layout
- c) Debug Layout
- d) Classic Undocked
- e) Classic Docked

3) Which view contains an outline of the components and classes used in a form?

E-Learning Series: Getting Started with Windows and Mac Development

- a) Project Manager
- b) Tool Palette
- c) Object Inspector
- d) Structure
- e) Debug Windows
- f) File Explorer

Answers to the Self Check Quiz:

1e, 2e, 3d