IBM Rational XDE Developer v2003.06.12 — .NET Edition

Evaluators Guide

Rational. software

Contents

Introduction

Welcome to the *IBM Rational XDE Developer v2003.06.12 — .NET Edition Evaluators Guide.* This guide has been created to help you evaluate IBM® Rational® XDE™ Developer and the unique benefits it brings to developers. Exercises in the guide will enable you to experience the product first-hand.

This guide is divided into the following sections:

- **Welcome to Rational XDE Developer** — An overview of Rational XDE Developer, including a summary of its key benefits and instructions for installing an evaluation copy of the software and getting started.
- Sections containing step-by-step exercises (each of which will take about 20 minutes to complete):
    - **Integrated Design and Development**
    - **Understanding Your Code**
    - **Developing Web Services**
    - **Development with Custom Patterns and Code Templates**

  The exercises are largely independent of each other; however, for the best introduction to Rational XDE Developer we recommend that you follow them sequentially.
- **Flexible Team Development and Configuration Management** — An overview of the integration between IBM Rational ClearCase® products and Rational XDE Developer, enabling you to distribute the development of models and code across your team based on your needs.
- **Rational RequisitePro Integration** — An overview of the integration between IBM Rational RequisitePro® and Rational XDE Developer, enabling you to manage software requirements documented in use cases.
- **Extending Your Development Experience** — An overview of the Rational Domain on IBM developerWorks, a Web site that provides a wealth of information for software professionals, including information about Rational XDE Developer and Microsoft® Visual Studio .NET technology.

We hope that you find this guide to be a convenient resource.

Welcome to Rational XDE Developer

Rational XDE Developer has been designed from the ground up as an extended development environment. Fully integrated into the Microsoft Visual Studio .NET integrated development environment (IDE), Rational XDE Developer enables developers to design and code within a single environment, avoiding the need to switch between different, nonintegrated tools.

If you are already familiar with Visual Studio .NET, you will find the Rational XDE Developer user interface very familiar. If this is your first exposure, you may want to visit the appendix of this guide to see an overview of the common windows you will be manipulating during the exercises.

Summary of Key Benefits

Rational XDE Developer enables you as a software developer to:

- **Do more in your IDE** — Traditional lifecycle support tools are only loosely coupled to your development environment; they have their own user interface, requiring you to use ALT+TAB to go back and forth between the two environments. This makes for an awkward and unproductive user experience. Rational XDE Developer offers a smoothly integrated environment that enables you to be much more productive.
- **Develop solid code faster** — You care about writing code: you want to do it fast, and you want it to be good, solid code. You know that visual modeling can help document and communicate your code's design, and that analyzing and debugging your code as you write it will make it more solid. But you're afraid that modeling and analysis tools will slow you down and get in the way of writing solid code. Rational XDE Developer doesn't get in your way; it helps you write better code faster. Its unique code template and patterns features enable reuse at both the code and model levels, which speeds up development, and its special "assisted modeling" capabilities let you create and edit Unified Modeling Language (UML) models in terms you know well from your IDE and its languages. Rational XDE Developer also offers runtime analysis tools that detect memory errors, highlight application performance bottlenecks, and identify untested code. These features help you find problems in your code and fix them before they surface in your customer's environment.
- **Develop more easily on a team** — Software development is a team sport. As a member of a software team, you have to deal with documentation, communication, requirements, version control, defect tracking, reporting, and overall process management. But you don't want these activities to get in your way — and Rational XDE Developer liberates you from these challenges. It integrates with Rational RequisitePro for viewing and managing ever-changing requirements. Additionally, Rational XDE Developer integrates with Rational ClearCase to give you version control features right inside your IDE, and with IBM Rational ClearQuest® to enable you to create defect reports instantly upon finding bugs when you analyze your code.

In addition:

- The Rational XDE Developer Plus edition is a superset of Rational XDE Developer that includes the Visual Trace capability and IBM Rational PurifyPlus. (The evaluation software is a copy of this edition.)
- The IBM Rational Unified Process® (RUP®) methodology offers best practices and a configurable process framework for guiding your development activities, and there is a special RUP configuration for .NET developers (as well as one for Java developers).
- A plug-in is available for Extreme Programming (XP) that offers lightweight best practices designed specifically for that development practice.

Installing Rational XDE Developer Plus

Before installing the evaluation software, first make sure your system complies with the following requirements:

Operating system	• Windows 2000 Professional, Service Pack 2 or Service Pack 3 (Service Pack 2 is recommended) • Windows 2000 Server, Service Pack 2 or Service Pack 3 • Windows 2000 Advanced Server, Service Pack 2 or Service Pack 3 • Windows XP Professional, Service Pack 1
Supported platform	Microsoft Visual Studio .NET 2003 (Microsoft Visual Studio .NET 2002 is also supported, but this guide assumes 2003 throughout.)
Processor	Minimum: Pentium III class, 500 MHz Recommended: Pentium III class, 1 GHz or higher
RAM	Minimum: 512 MB Recommended: 1 GB
Disk space	Minimum: 500 MB for installation directory, 100 MB for workspace Recommended: 2–5 GB for workspace
Video (screen resolution)	Minimum: 800 x 600 pixels, 256 colors Recommended: 1024 x 768 pixels, 16-bit color

Follow these steps to install and start the evaluation copy of Rational Rose XDE Developer Plus:

1. If you are installing from the IBM Software Development Platform – Software Evaluation Kit (SEK) Windows platform DVD:
    a. In the **Design and Construction** section on the main user interface:
        i. Click **WebSphere** to get to the Rational XDE Developer Plus Java evaluation
        ii. Click **Microsoft .NET** to get to the Rational XDE Developer Plus .NET evaluation
        iii. Click **Other Leading IDEs** to get to the Rose Enterprise evaluation
    b. Within each of these child user interfaces, click the **Evaluation installation** link on the left navigation bar to download the self-contained executable product evaluation package and save the .exe to a temporary directory.
    c. Within each of these child user interfaces, click the **Evaluation registration** link on the left navigation bar to go to the ibm.com website to register for your product evaluation license key. Fill out the online form and follow the steps that will lead to the product evaluation page. This page provides you with the ability to download the product evaluation (which you should ignore, if you downloaded the evaluation from the DVD), and access to the product evaluation license key.
    d. Copy the license key to the folder in which you installed the product evaluation.
2. If you are installing Rational Rose XDE Developer Plus from a Web download:
    a. Download the self-contained executable product evaluation package and save the .exe to a temporary directory.

b. Double-click the .exe package and begin the installation.

c. Don't forget to copy the license key from the product evaluation download web page to the folder in which you installed the product evaluation.

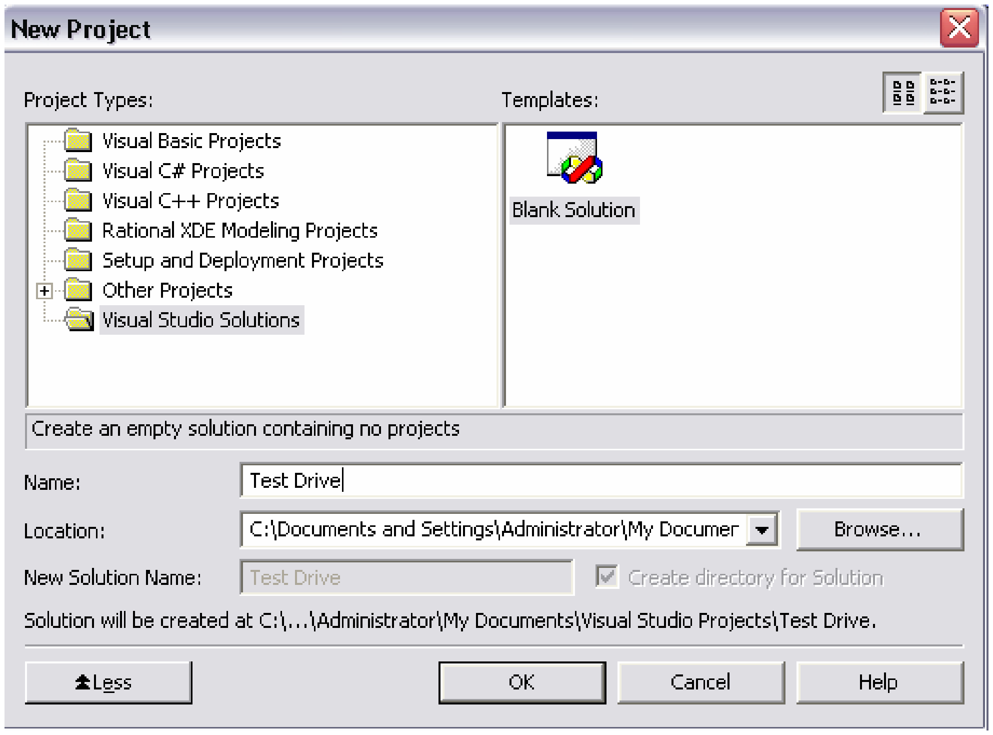
Getting Started

1. Click **Start > Programs > Microsoft Visual Studio .NET 2003 > Microsoft Visual Studio .NET 2003** to start Visual Studio .NET and Rational XDE Developer Plus.

2. Click **File > New > Blank Solution**. Keep the default selections: **Visual Studio Solutions** as the project type and **Blank Solution** as the template.

   This Visual Studio .NET blank solution will hold the projects created in the subsequent exercises.

3. Name the solution `Test Drive` and accept the default location. Click **OK**.



Create an empty solution containing no projects

Name: Test Drive

Location: C:\Documents and Settings\Administrator\My Documer

New Solution Name: Test Drive

Create directory for Solution

Solution will be created at C:\...\Administrator\My Documents\Visual Studio Projects\Test Drive.

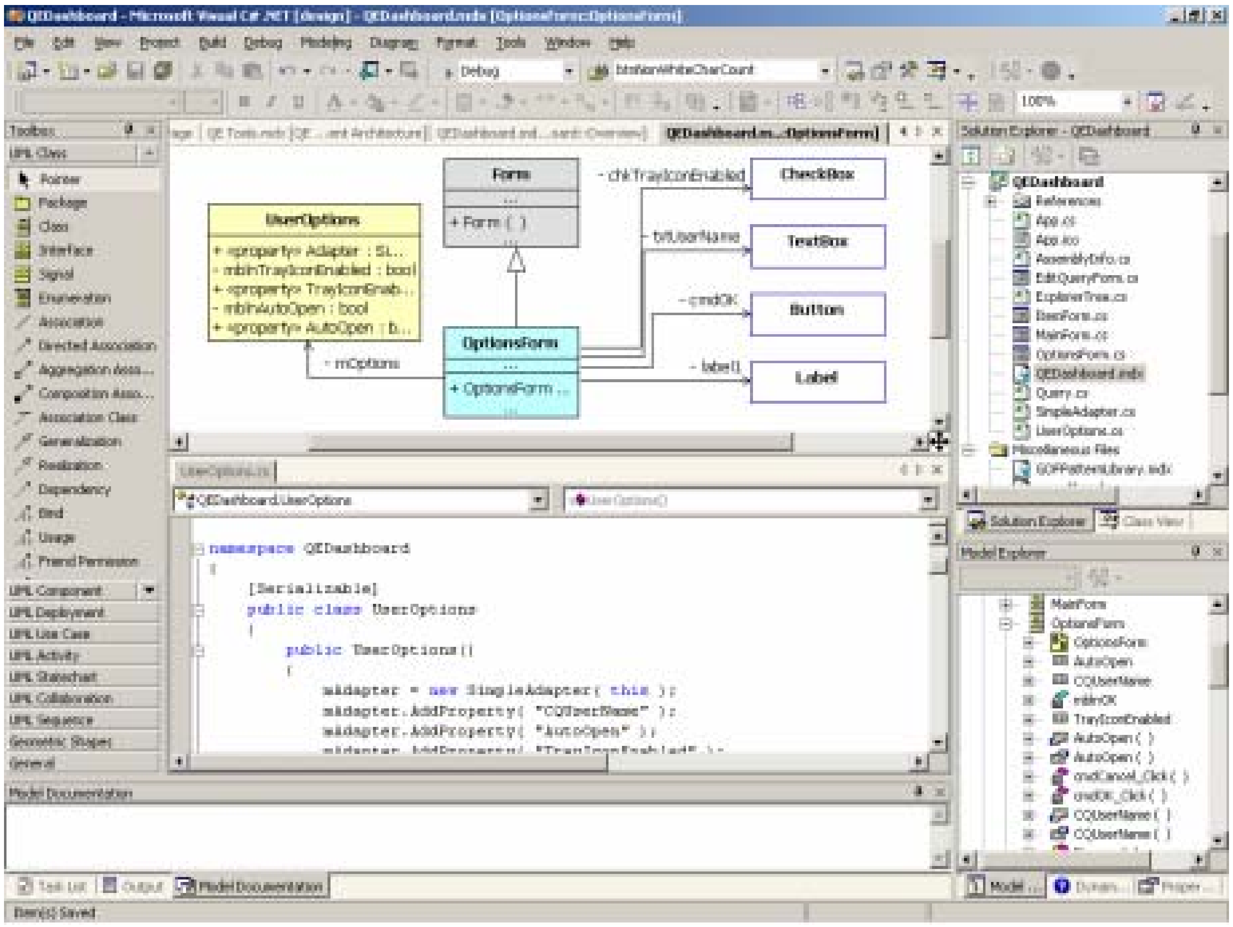
You are now ready to experience the power of Rational XDE Developer Plus, using the `Test Drive` solution as your starting point for the exercises in this guide.

Integrated Design and Development

Rational XDE Developer lets you combine your design and your development into a seamless, tightly integrated experience. This extended development environment provides essential developer capabilities that are fully integrated into the Microsoft Visual Studio .NET technology, thereby providing a consistent look and feel.

Modeling capabilities are now as much a part of your IDE as your code editor, compiler, and debugger. With this combined environment, your primary development tools all follow the same menus, gestures, and usage metaphors, accelerating your learning curve and promoting the use of design, code generation, and code synchronization as a daily function. Ultimately, this leads to developing better software, faster.

With Rational XDE Developer, you model using the industry-standard Unified Modeling Language (UML). With this common language, communication is better, development time is shorter, complex systems are easier to understand, and designs are cleaner and easier to maintain. Rational XDE Developer automatically generates the UML representation of your code directly from your source files. The generated models are beneficial for documenting your work, understanding complex projects through visualization, and harvesting your work for future reuse.



```
namespace QEDashboard
{
    [Serializable]
    public class UserOptions
    {
        public UserOptions()
        {
            mAdapter = new SingleAdapter( this );
            mAdapter.AddProperty( "CQUserName" );
            mAdapter.AddProperty( "AutoOpen" );
```

Figure 1. Design and Development Within One Environment — Developers can access the code and the model simultaneously.

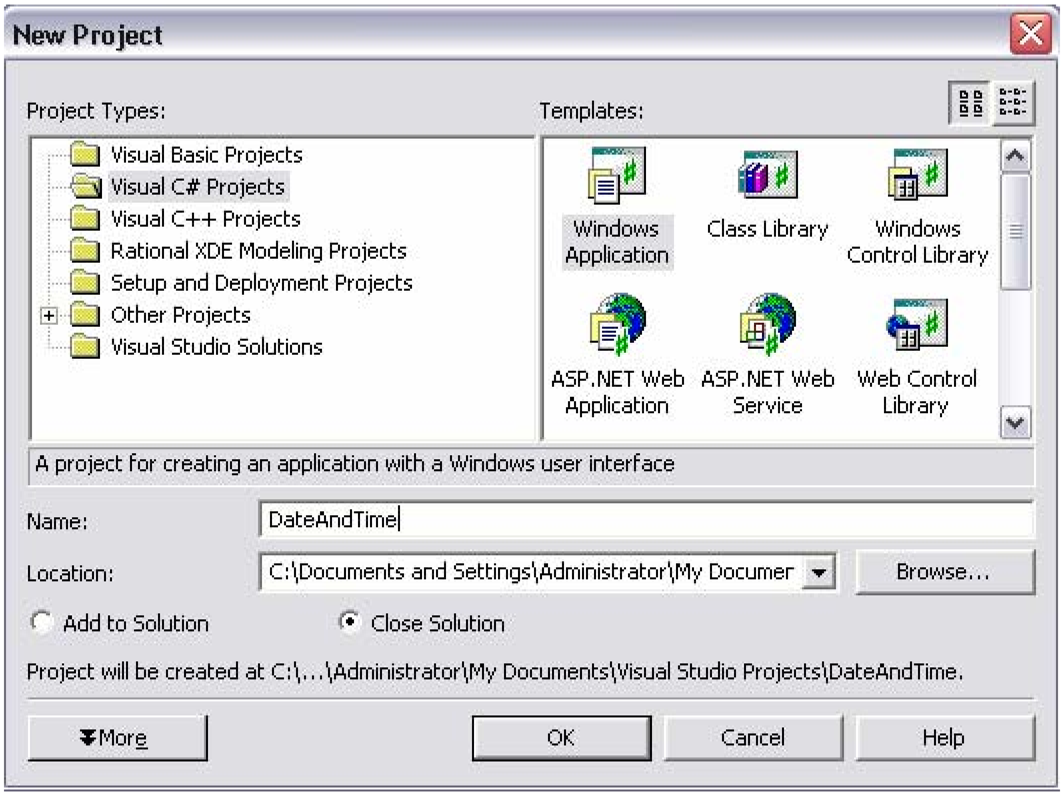
Experience *Integrated Design and Development*

In this section, you will gain experience with the following Rational XDE Developer features:

- Creating a model for a new Visual Studio .NET project
- Automatic or manual code synchronization
- Modeling references to the .NET Framework
- Visualizing code using the Unified Modeling Language

While creating a simple C# application, you will see Rational XDE Developer build the visual model for the application, and you will use this model to understand how the application fits within the Microsoft Visual Studio .NET Framework. Along the way, you will experience using Rational XDE Developer to quickly change properties of multiple elements. Finally, you will see automatic synchronization in action by modifying your model and watching your code update, and then by modifying your code and watching your model update.

1. If you have experimented with auto-synchronization before reaching this exercise, do the following to check that it is turned off (as assumed by this exercise): click **Tools > Options** and, on the left in the dialog box, expand the **Rational XDE** folder and then **Round-Trip Engineering** to see the **Auto Synchronization** options; make sure **Automatic Synchronization** is not selected. Click **OK**.
2. With the `Test Drive` solution open, click **File > New > Project**.
3. Select **Visual C# Projects** as the project type and **Windows Application** as the template.
4. Rename the project as `DateAndTime` and keep the default location.
5. Verify that the **Add to Solution** button is selected. Click **OK**.



A project for creating an application with a Windows user interface

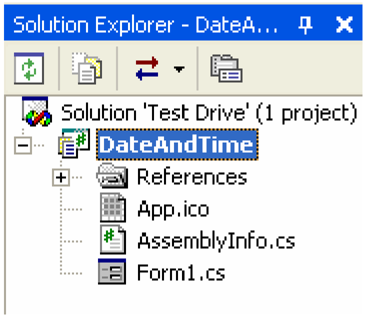
Name: DateAndTime

Location: C:\Documents and Settings\Administrator\My Documer

Add to Solution [✓ Close Solution]

Project will be created at C:\...\Administrator\My Documents\Visual Studio Projects\DateAndTime.

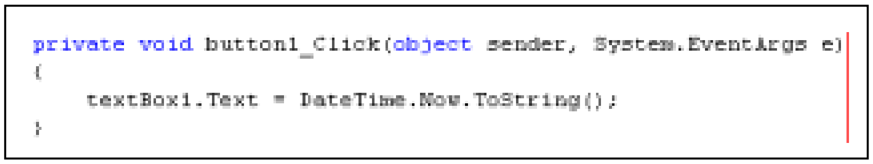
The `DateAndTime` project is added to the `Test Drive` solution.


Solution Explorer - DateA...
Solution 'Test Drive' (1 project)
DateAndTime
References
App.ico
AssemblyInfo.cs
Form1.cs

6. Click the **Toolbox** button located on the left side of the Visual Studio .NET IDE. (If the **Toolbox** button is not visible, click **View > Toolbox**.)
7. Click the **Button** tool ab| Button and then click in Form1 where you want to place the control. Repeat this step for a text box abl TextBox. Leave the default names (button1 and textBox1).
8. Double-click button1 to open the source code window for the button.
9. Complete the button1_Click method with the following call:

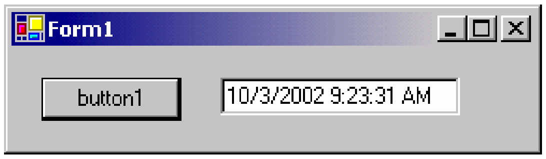
   textBox1.Text = DateTime.Now.ToString();

   The method should look like this:



```
private void button1_Click(object sender, System.EventArgs e)
{
    textBox1.Text = DateTime.Now.ToString();
}
```

10. In the Solution Explorer, right-click the DateAndTime project and click **Debug > Start new instance** to build and execute the application.
11. Verify that the application displays the date and time in textBox1 when you click button1.

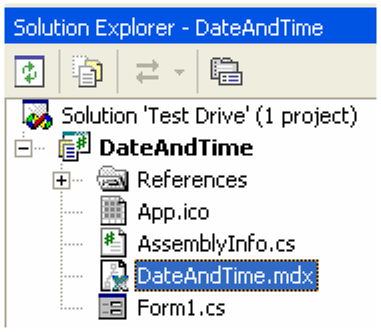


12. Quit the application by closing the Form1 window.

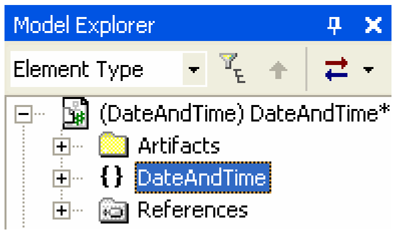
Creating UML Models from Code

13. In the Solution Explorer, select the DateAndTime project and click the **Synchronize** button .

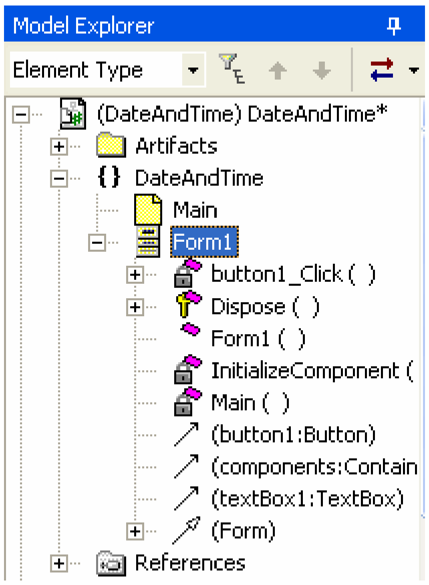
   A Rational XDE Developer model, DateAndTime.mdx, is added to the DateAndTime project.



14. Click the **Model Explorer** tab and notice the `DateAndTime` model.



15. Expand the {} DateAndTime namespace and the `Form1` class to view the methods, attributes, and UML associations for the `Form1` class created during synchronization.

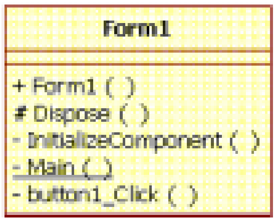


Key Benefit

Rational XDE Developer and Visual Studio .NET work together using shared technology and tight integration to enable you to create a UML design model of your application with just one click from within the Visual Studio .NET IDE. You can visually model your code without ever leaving your favorite development environment, making designing your applications easier than ever.

Visualizing the .NET Framework Classes

16. From the Model Explorer, drag the `Form1` class (Form1) onto the drawing surface, where the `Main` diagram should already be opened.



17. Right-click `Form1` in the diagram and click **Add Related Shapes** (Add Related Shapes...).

    This adds to the current diagram additional model elements that are related to the element selected in the diagram.

18. In the Add Related Shapes dialog box, set **Select in Model(s)** to **All Models**. Keep the **Expands to N Levels** setting at **1**. Click **OK**.

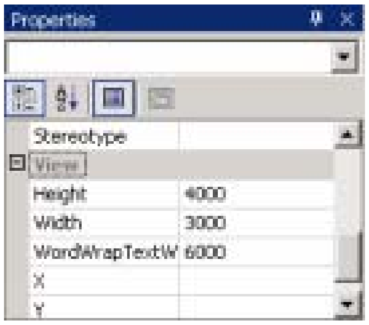
Key Benefit

Through a visual model, Rational XDE Developer helps you learn about the .NET Framework and understand how your application uses the .NET Framework components. By expanding your classes, you can quickly view dependencies between your application and specific .NET Framework components. This information gives you access to the system library details you need while building your application.

Rational XDE Developer lets you refine the search criteria when adding related shapes to your diagram. You can add related shapes based on relationship type and the model in which they reside. This level of flexibility enables you to easily visualize your code at just the right level of abstraction.

19. Click **Edit > Select All Shapes** to select all the shapes in the diagram.

20. In the Properties window (Properties, located at the bottom right), scroll down to the **View** section and enter 4000 for `Height` and 3000 for `Width`. Click elsewhere to make the diagram reflect the change.

Height	4000
Width	3000
WordWrapTextW	6000
X	
Y	

    Note that only the properties common to all selected elements are displayed.

Key Benefit

Rational XDE Developer's tight integration with Visual Studio .NET provides a seamless way to modify properties of multiple model elements in one gesture, the same gesture that you use when you develop your application.

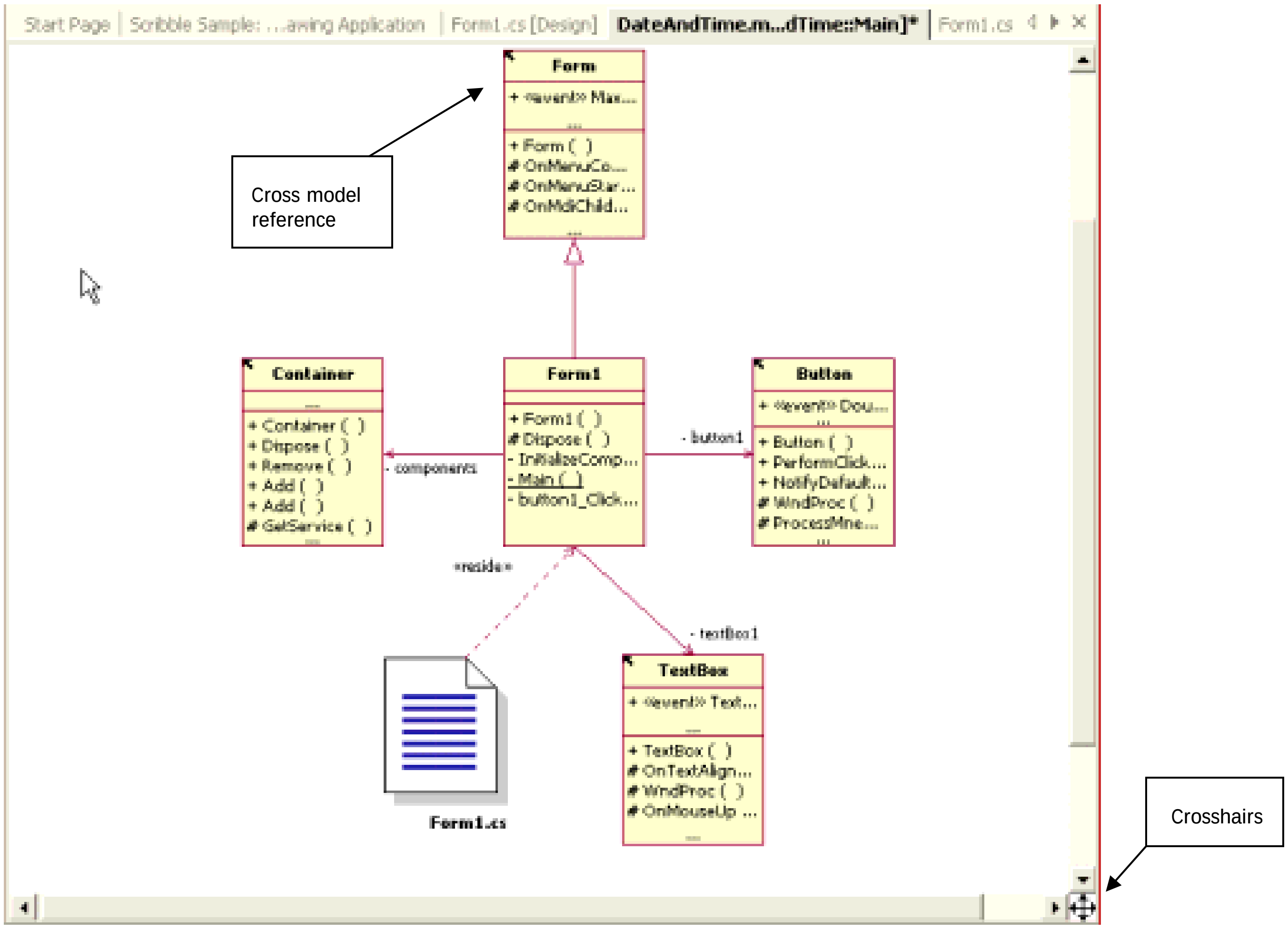
21. Click the diagram tab DateAndTime.m...dTime::Main]* to give the focus to the diagram.

22. On the toolbar, click the **Arrange All Elements** button and change the **Zoom** factor to 75% 75% for better viewing.

Key Benefit

A number of mechanisms are available for arranging and viewing your diagrams, saving you time and effort while maximizing the value of your model information.

23. Use the crosshairs in the bottom-right corner of the diagram to navigate the diagram with a bird's-eye view.



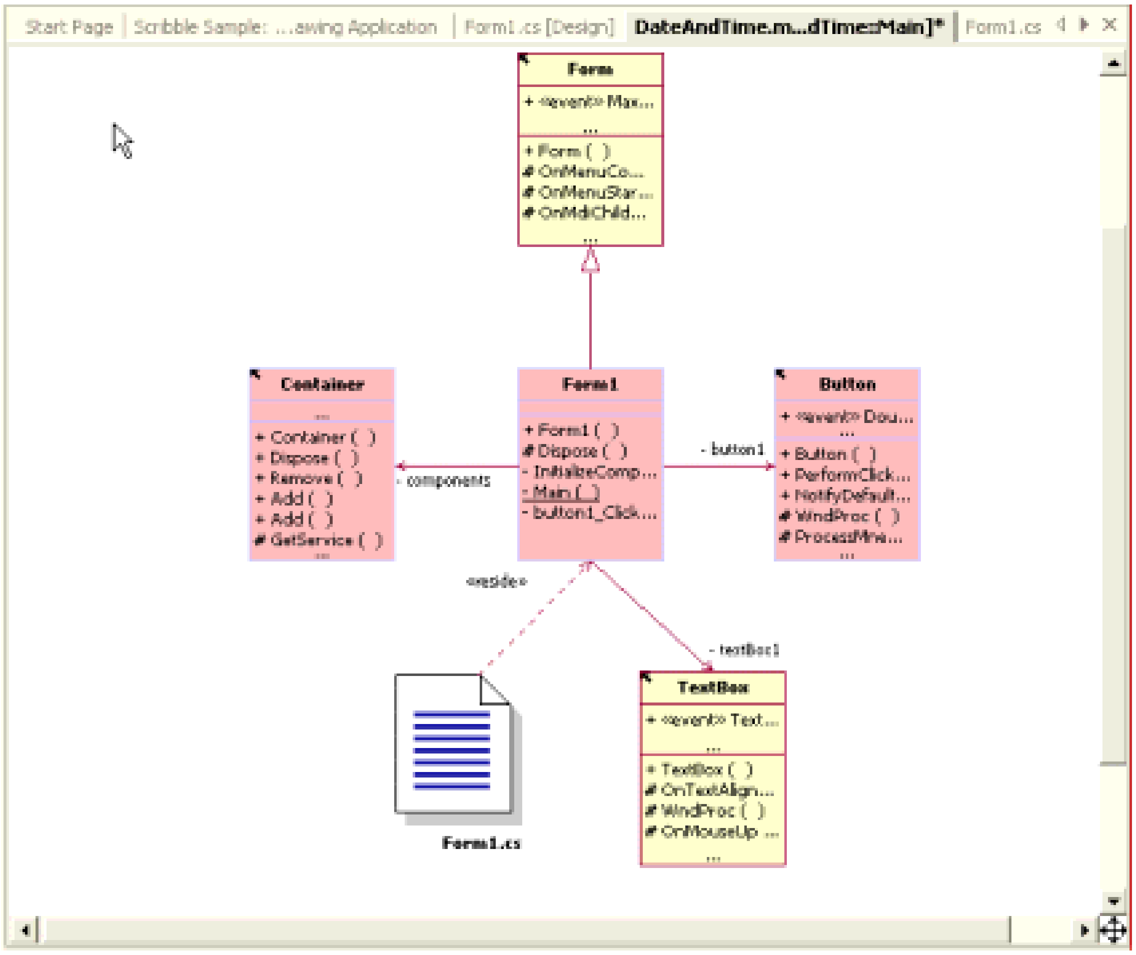
Key Benefit

Rational XDE Developer goes beyond the ability to store designs in multiple models; it maintains live references across models. Cross-model references let you remain in your current model and make updates to elements located in other models, eliminating duplicate work. This also gives you the freedom to organize your application development efforts to best suit your team's needs. References across models are indicated visually with a black arrow icon in the upper-left corner of the model, for easy identification. Full search and replace capabilities across models ensure that you can maintain model integrity and consistency without added effort.

24. Using the CTRL key, select three of the classes displayed in the `Main` diagram.

25. Click the fill color button on the main toolbar and select a new color.

    The background of the selected classes changes to that color.



Key Benefit

Rational XDE Developer provides formatting options, such as colors, that you can use to make your diagrams easier to read and understand.

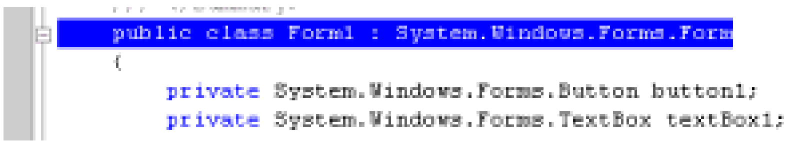
Viewing Code and Model Simultaneously

26. At the top of the drawing area, click the **Form1.cs** tab, which represents the code view for the `Form1` class.
27. Drag the tab to the bottom of the IDE (the mouse pointer will resemble a document as you do this); then release the mouse button.

Key Benefit

You can customize your work area to see both the visual model and your code at the same time, enabling you to work on either the code or the model and see how your changes affect each area.

28. With `Form1` selected in the diagram, click **Format > Auto Resize** to better view the text of attributes and operations.
29. Right-click `Form1` in the `Main` diagram and click **Browse Code** to view the code for that form in the Visual Studio IDE code editor window.



```
public class Form1 : System.Windows.Forms.Form
{
    private System.Windows.Forms.Button button1;
    private System.Windows.Forms.TextBox textBox1;
```

Key Benefit

Rational XDE Developer conveniently takes you to the location in the code that applies to your selection context.

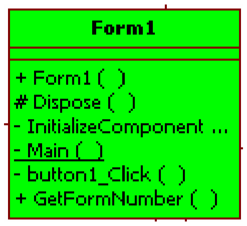
Synchronizing Code and Model Automatically

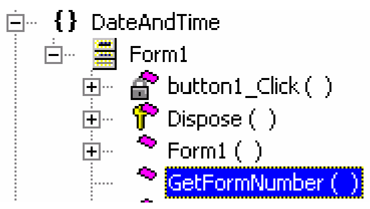
30. Click **Tools > Options**. On the left, scroll down, expand the **Rational XDE** folder, and then expand **Round-Trip Engineering** to see the **Auto Synchronization** options.
31. Select the **Automatic Synchronization** check box. Keeping other settings at their defaults, click **OK**.

Key Benefit

You can leave the burden of keeping your code and model in sync to Rational XDE Developer. Turning auto-synchronization on will save you the hassle of manually synchronizing your code with your model. Additionally, Rational XDE Developer lets you decide when synchronization should occur and how conflicts between code and model will be handled. Because auto-synchronization upon each and every change can be distracting, you can turn it off until you are ready to generate code from your model.

32. In the Main diagram, right-click the Form1 class and click **Add UML > Operation**.
33. Name the new operation GetFormNumber. Click away from the class and notice that the Model Explorer displays the new operation.



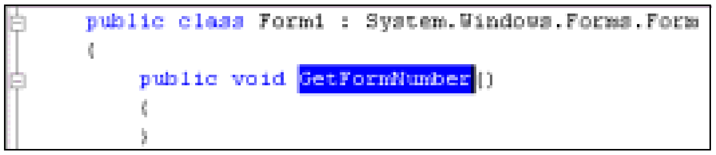


Key Benefit

Rational XDE Developer provides multiple representations of model elements and keeps them in sync at all times.

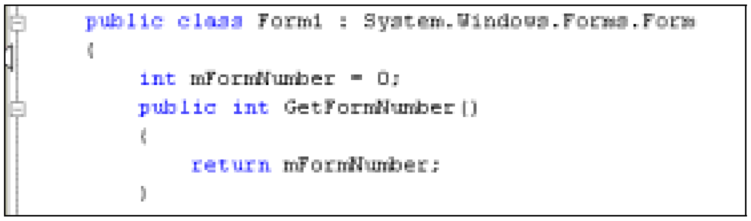
34. With Form1 selected in the diagram, click **Format > Signature > Operation Signature** to display the operation signatures in the diagram.
35. On the toolbar, click the **Arrange All Elements** button.
36. Click the **Form1.cs** tab.

    Because auto-synchronization is on, this action automatically updates the Form1 code with the new GetFormNumber operation.



```
public class Form1 : System.Windows.Forms.Form
{
    public void GetFormNumber()
    {
    }
```

37. Modify the source code as follows:

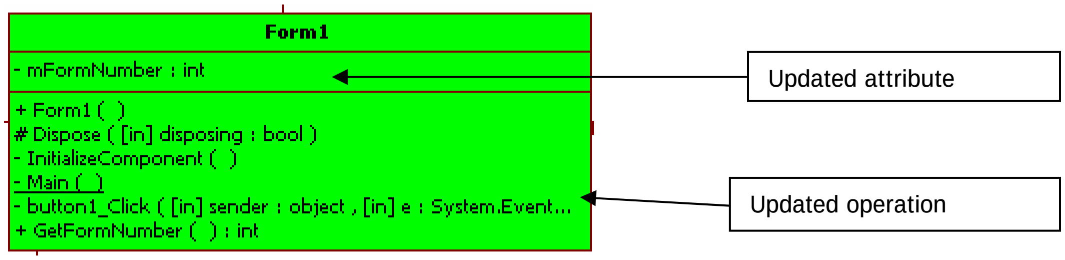


```
public class Form1 : System.Windows.Forms.Form
{
    int mFormNumber = 0;
    public int GetFormNumber()
    {
        return mFormNumber;
    }
```

38. Click in the `Main` diagram.

    Because auto-synchronization is on, the synchronization between code and model is automatically initiated, as indicated by the pulsing Synchronize icon on the Rational XDE Developer status bar.

39. Wait until the pulsing icon on the status bar disappears, indicating that the synchronization is complete. Once the code and model are synchronized, you can see that the changes you made to the `Form1` class were automatically updated in the model file.



```
Form1
- mFormNumber : int
+ Form1 ( )
# Dispose ( [in] disposing : bool )
- InitializeComponent ( )
- Main ( )
- button1_Click ( [in] sender : object , [in] e : System.Event...
+ GetFormNumber ( ) : int
```

Key Benefit

With auto-synchronization turned on, changes to your source code are automatically updated in your model. Because Rational XDE Developer keeps the visual representation of your application current, you can work where you feel most comfortable, either in the model or in the code, without the hassle of having to manually synchronize the two.

40. In the Solution Explorer, right-click the `DateAndTime` project and click **Save DateAndTime**.

41. Close the **DateAndTime....ndTime::Main]**, **Form1.cs**, and **Form1.cs [Design]** windows.

Benefits of *Integrated Design and Development*

Having integrated design and development increases your productivity by providing an extended development environment in which the code and the model can be kept synchronized at all times. By handling code-model synchronization for you, Rational XDE Developer lets you work where you feel most comfortable, either in the model or in the code. You simply specify synchronization rules to resolve any conflicts between code and model, and Rational XDE Developer handles the rest.

Additionally, Rational XDE Developer maximizes your efficiency by letting you decide *when* to turn on auto-synchronization. When you want to test a few designs or apply some patterns without committing the model changes to code, you can turn off auto-synchronization. Other times, when you don't want to have to think about whether your code and model are in sync, you can turn on auto-synchronization, and any change in the model or code will automatically be synchronized.

The tight, seamless integration of Rational XDE Developer with Visual Studio .NET makes modeling capabilities as much a part of the IDE as the code editor and debugger. You can quickly visualize your code in UML models to gain a better understanding of the relationships and dependencies between the code and the .NET components it uses. Rational XDE Developer provides flexible options for expanding, arranging, and navigating model elements, so you can instantly visualize specific areas of interest. You can use this same functionality to manipulate visual models of the .NET assemblies provided by Rational XDE Developer, further accelerating your .NET Framework learning curve.

Understanding Your Code

A very real challenge for developers is maintaining, debugging, and enhancing code they may not have written. Typical visual modeling tools use UML to visually describe the static aspect of code, classes, methods, attributes, relationships, and so on. In most cases this static information is not sufficient; developers need to understand *how* the application behaves. Typically, the best approach to understanding application behavior has been a tedious one of strategically setting breakpoints and stepping through the code. This approach can be time-consuming and difficult, especially for the event-driven code that is typical of user interface development.

Rational XDE Developer Plus facilitates the process of diagnosing and understanding application behavior with Visual Trace, a capability that visualizes program execution for run-time analysis. Object creation, deletion, and interaction are all completely evident. Visual Trace records object information only for classes specified by the user, making it easy to pinpoint specific program behavior. Since Visual Trace captures only the code actually executed, the resulting trace diagram is not cluttered with every conditional path through the code. Trace diagrams provide a way to automatically document and communicate program behavior with the rest of the team.

Experience *Understanding Your Code*

In this section, you will gain experience with the following Rational XDE Developer Plus feature:

- Visual Trace

Multiple Document Interfaces (MDIs) are the standard user-interface approach to many window-based applications, the most notable being word-processing applications that allow the user to open and modify multiple documents at the same time. In this exercise, you will use the Visual Trace capability of Rational XDE Developer Plus to understand how the `Scribble` application, a simple MDI, works. You will use the Visual Trace capability to identify specific program behavior, such as how to create a new document and view, how documents interact with multiple views, and how main menu settings, such as changes to the `Scribble` pen, affect individual child windows.

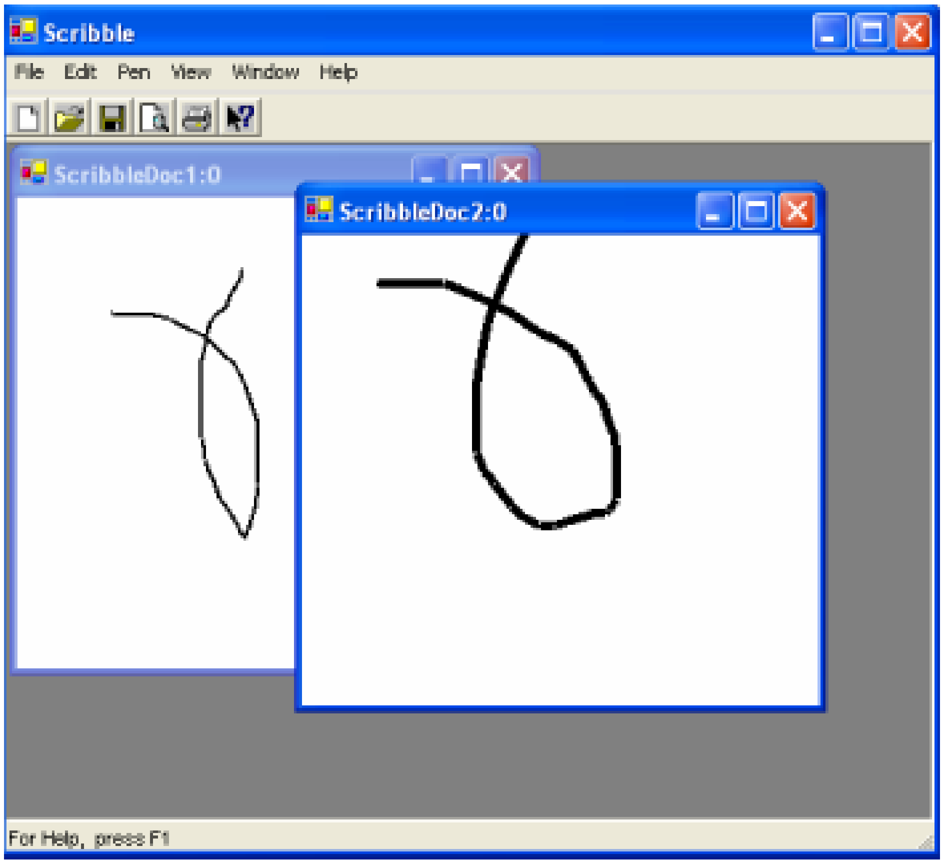
Loading and Running the Scribble Application

Note that, because the Visual Studio .NET samples do not typically get installed when you install Visual Studio .NET, you will likely be asked for your Visual Studio CD (or a network location from which you installed Visual Studio .NET) at the start of the exercise.

1. On the Start page, click the **Online Resources** tab. If the Start page is not visible, click **Help > Show Start page**.
2. Click **Get Started** in navigation bar on the left.
3. Click **Visual C# Developer** in the **Samples Profile** drop-down list.
4. Keep the **Keyword** default value selected for the **Filter by** field. In the **Filter by** text box, type `scribble` and click **Go**.
5. Click the Scribble Sample: Visual C# MDI Drawing Application:Visual C# Samples search result. You may be asked to locate the sample solution file, in which case you can either load your Visual Studio .NET CD or point to the network location from which you installed.
6. Click Copy All Files and accept the default location of `My Documents`.

   Note: In this exercise, you copy the sample files, rather than load them, to remain within one instance of Visual Studio .NET. Loading the files would invoke another instance of Visual Studio .NET, and running multiple instances of Visual Studio .NET can become memory-intensive.

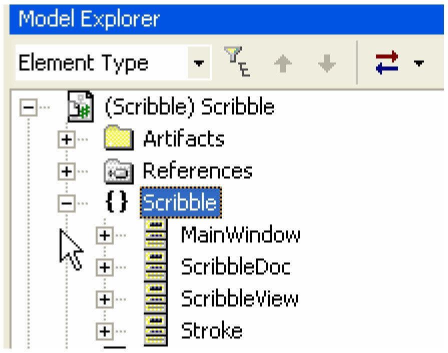
7. Click **File >** **Open Solution** and navigate to the MyDocuments\samples\Visual C# .NET 2003\General\Scribble\ directory.
8. Select Scribble.sln and click **Open**. If asked to save changes to the Test Drive solution, click **Yes**.
9. Press F5 to build and run the Scribble application.
10. Drag the mouse within the ScribbleDoc1:0 window to draw.
11. On the main **Scribble** menu, click **File > New** to create a new document, ScribbleDoc2:0.
12. On the main menu, click **Pen > Thick Line** to enable a thicker drawing line.
13. Drag within the ScribbleDoc2:0 window to draw.
14. Click other menu options to familiarize yourself with the application.
15. On the main menu, click **File > Exit** to close the Scribble application. A dialog box will prompt you to save your changes; click **No**.



Creating a UML Model from Code

16. In the Solution Explorer, select the Scribble project Scribble and click the **Synchronize** icon .

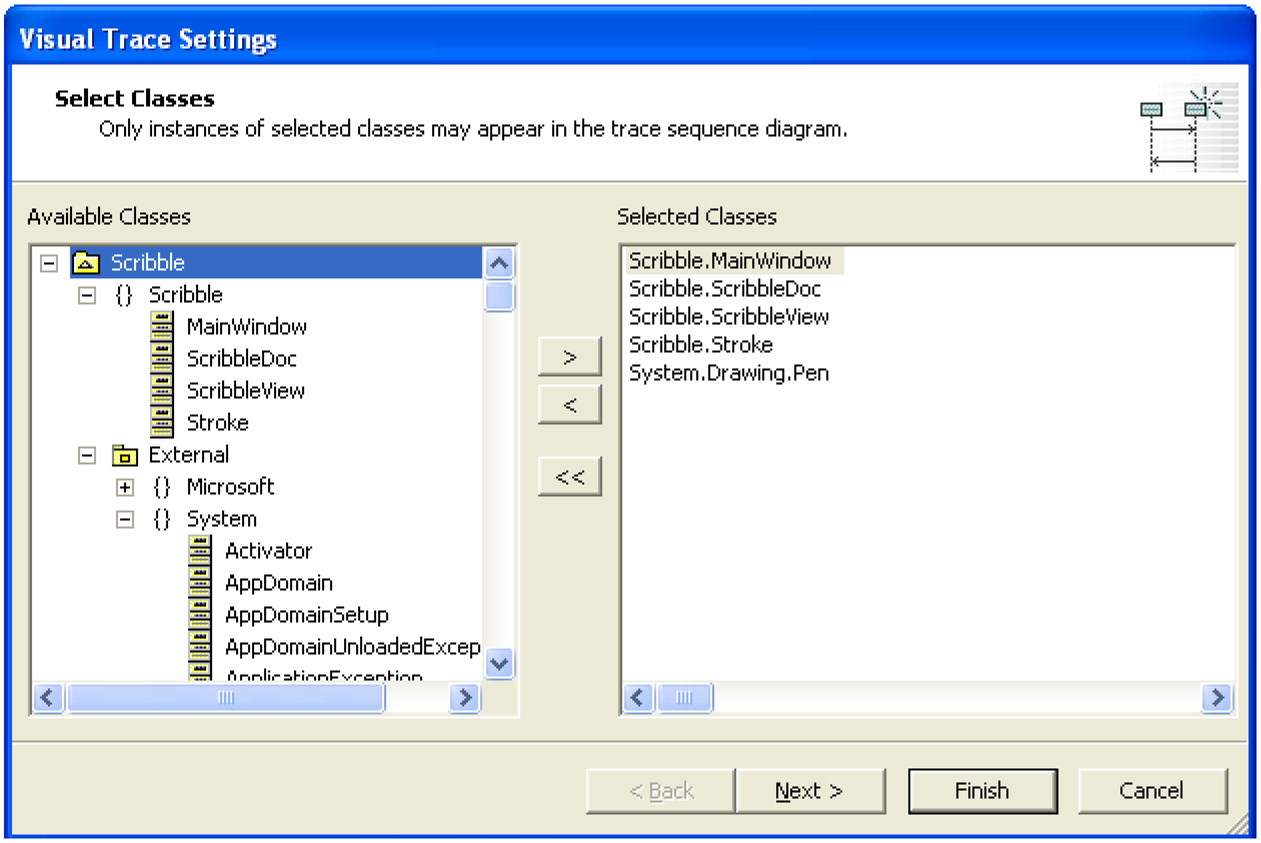
    Rational XDE Developer will synchronize the entire project and create a new UML model from the code.
17. In the Model Explorer, locate and expand the Scribble namespace to view the classes recognized during synchronization.



Visually Tracing the Scribble Application

18. On the main menu, click **PurifyPlus > Visual Trace > Engage Visual Trace**.
19. In the Visual Trace Settings dialog box, expand the `Scribble` folder and the `Scribble` namespace.
20. Add the four classes displayed in the `Scribble` namespace to the list of selected classes.

    These are the same four classes you just viewed in step 17.
21. Expand the `External` package and add the `System.Drawing.Pen` class.


Visual Trace Settings

Select Classes

Only instances of selected classes may appear in the trace sequence diagram.

Selected Classes

Scribble.MainWindow
Scribble.ScribbleDoc
Scribble.ScribbleView
Scribble.Stroke
System.Drawing.Pen

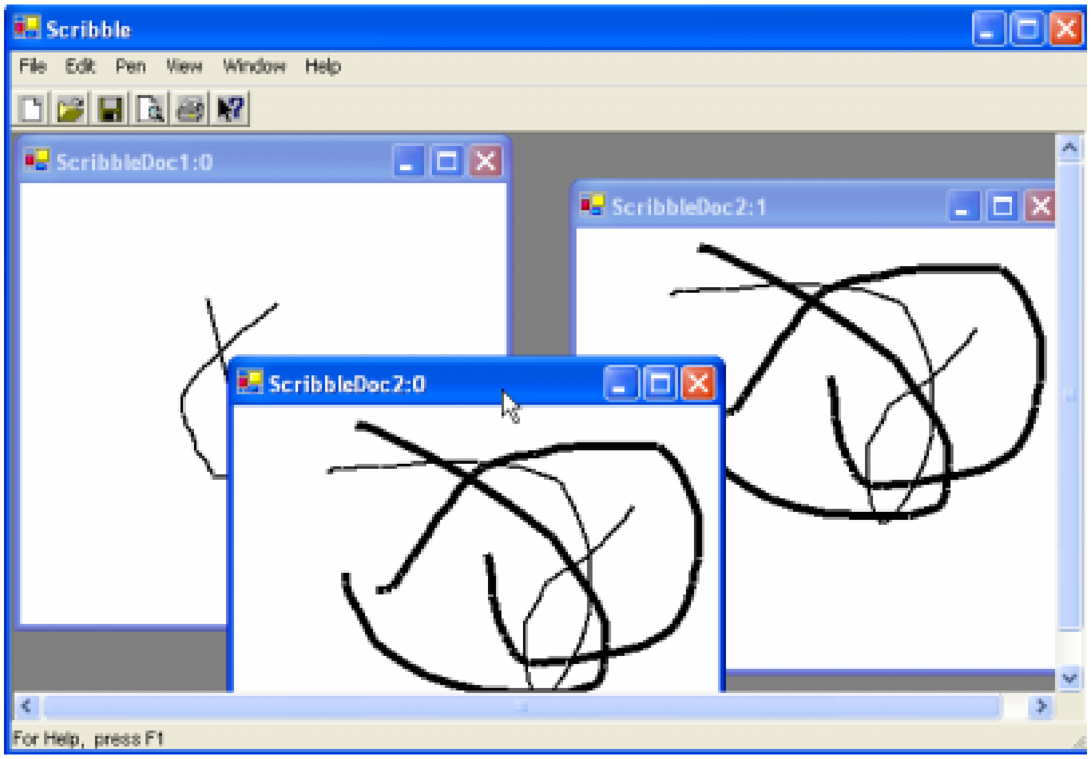
22. Click **Finish**.

Key Benefit

The Visual Trace capability of Rational XDE Developer Plus enables you to identify program behavior quickly by capturing trace data for only the classes you specify. You can even specify .NET Framework classes used by your application to understand how your application interacts with those components.

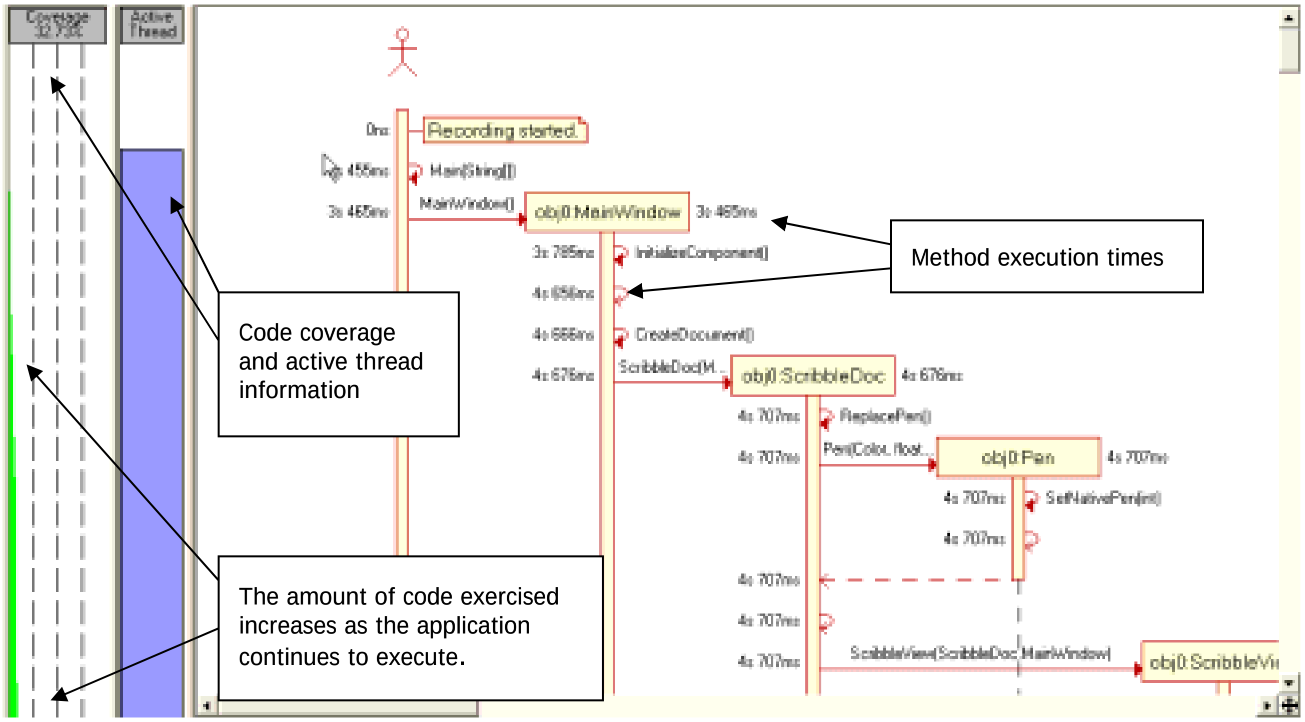
23. Select **PurifyPlus > Visual Trace > Engage Visual Trace** to start the trace.
24. Press Ctrl F5 to build and run the Scribble application again. Watch the Visual Trace diagram build as the application runs.
25. Drag within the ScribbleDoc1:0 window to draw.
26. On the main menu, click **File > New** to create a new document, ScribbleDoc2:0.
27. Drag within the ScribbleDoc2:0 window to draw.
28. On the main menu, click **Pen > Thick Line**.
29. On the main menu, click **Window > New Window** to create a new view of ScribbleDoc2:0, labeled ScribbleDoc2:1.

    The naming convention, ScribbleDoc*document-number*:*view-number*, distinguishes documents and their views. In this case ScribbleDoc2 has two views, view 0 and view 1.
30. Position ScribbleDoc2:1 and ScribbleDoc2:0 so that you can see them simultaneously.
31. Drag within the ScribbleDoc2:1 window to draw. Note how both windows are updated, yet ScribbleDoc1:0 is not.



    Views pointing to the same document are all updated when the document changes, even if they are not the active view.
32. On the main menu, click **File > Exit**. Do not save your changes.

Congratulations! You have just created your first trace diagram.

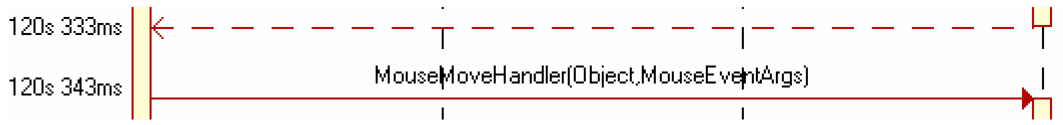


Key Benefit

Visual Trace diagrams provide a quick, visual way for you to perform run-time analysis and assess the health of your application. Code coverage information is captured for the classes under trace, indicating the percentage of code exercised during the trace. Method execution time is displayed adjacent to each method, making it easy to identify potential application bottlenecks. For multithreaded applications, the active thread is color-coded so that you can visually determine the functions executed per thread.

Manipulating Your Trace Diagram

33. Scroll to the top of the trace diagram and slowly scroll down until you locate the MouseMoveHandler(Object,MouseEventArgs) message.



34. Right-click the message and click **Filter Message** on the context menu.
35. Repeat this for the PaintHandler(Object,EventArgs), GetCurrentPen(), Min(int32,int32), and Max(int32,int32) messages.

Key Benefit

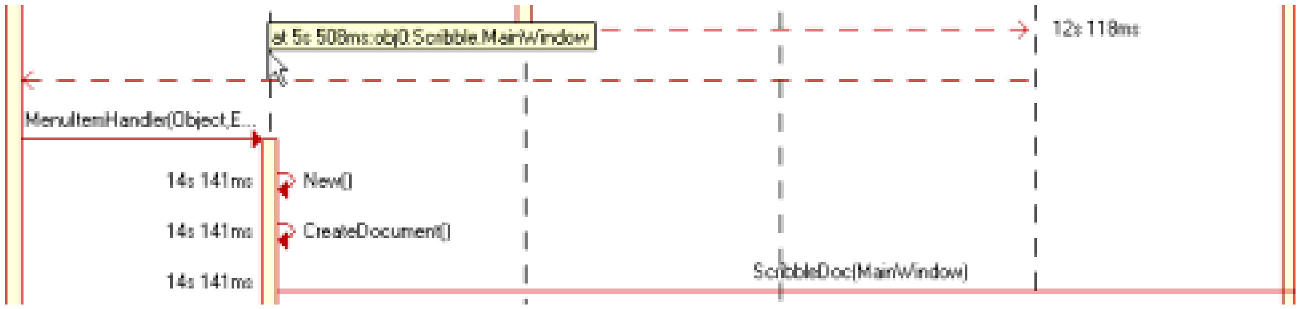
You can easily filter specific messages that are not relevant to the program behavior you are trying to capture and diagnose. To facilitate iterative debugging, you can store several messages in a single filter and just apply the filter to a later trace diagram.

Identifying Application Behavior with Visual Trace: Creating New Documents

During the trace, you created multiple documents (ScribbleDoc1:0 and Scribble2:0) and a single document with multiple views (Scribble2:0 and Scribble2:1). Using the trace diagram, you will be able to identify and understand how documents and views communicate.

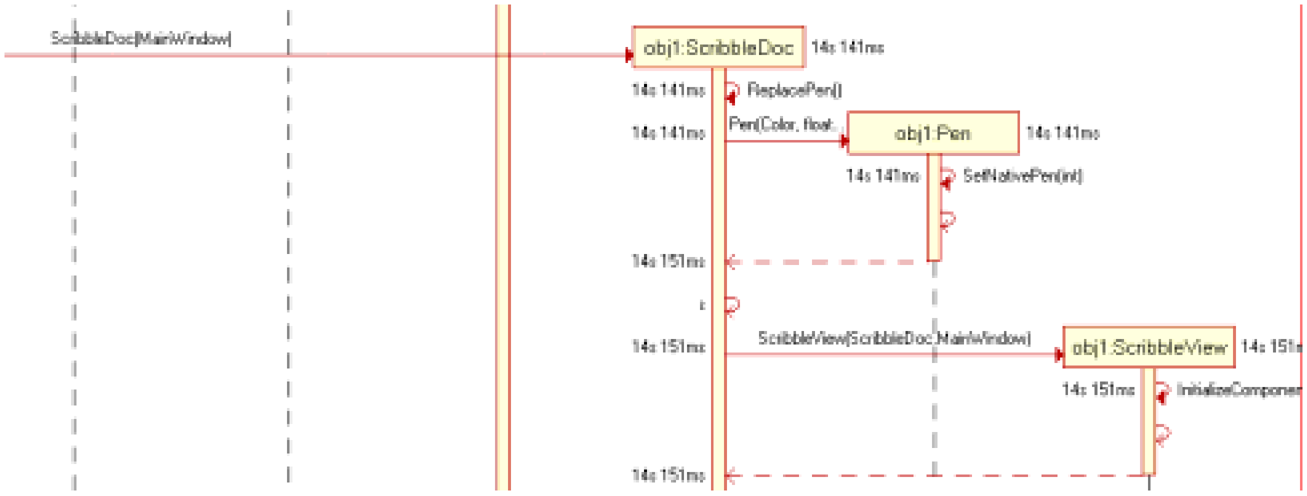
36. From the top of the trace diagram, scroll down until you can identify where obj0:MainWindow handles a menu event. Here you can see how the MainWindow class creates a new ScribbleDoc object.

    Hint: Holding your mouse button down on a lifeline will display the object and elapsed time at that point in the lifeline.

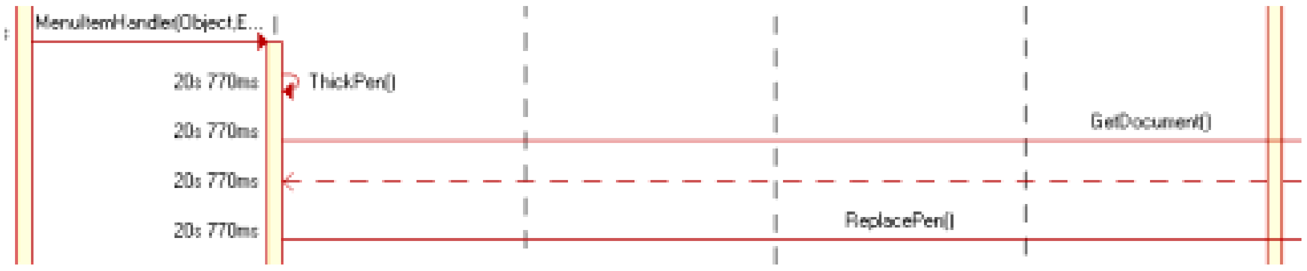


37. Scroll down slightly and follow the ScribbleDoc(MainWindow) message to the right. You can see obj1:ScribbleDoc create a new pen object, obj1:Pen, associated with the document, and then create a new view associated with the document.

    Since a pen appears to be associated with a specific document, we can deduce that two documents can have different pen styles and that switching between documents will maintain the appropriate pen selection.

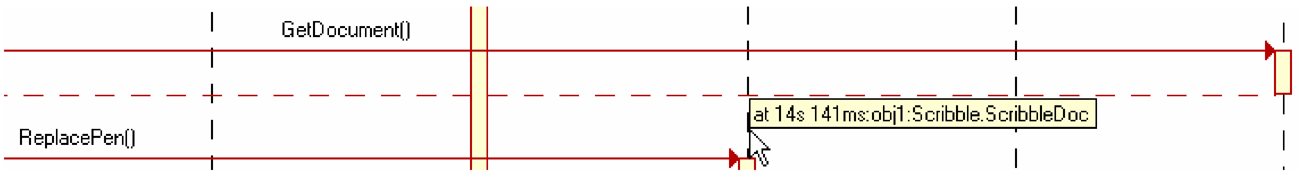


38. Scroll further down in your trace diagram to where the obj0:MainWindow is notified that a ThickPen menu event has occurred. Here your deductions are confirmed.



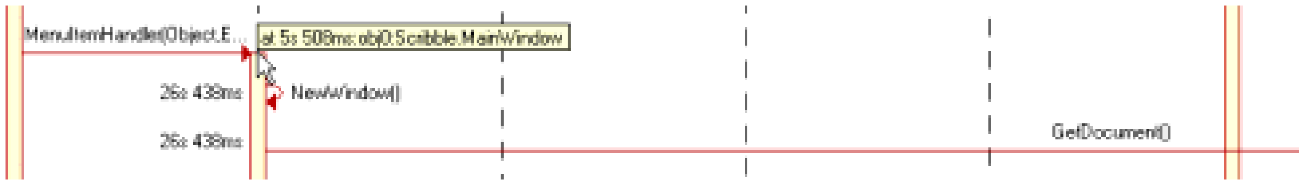
39. Scroll slightly to the right. You can see obj0:MainWindow obtain the current document from the active ScribbleView view, obj1:ScribbleView. Then obj0:MainWindow calls the ReplacePen function

for the current document. Using the call stack would be one way to determine the value of the current document. Using the trace diagram, you can determine the current ScribbleDoc document associated with the active ScribbleView view by following the ReplacePen message. Here you can see it is obj1:ScribbleDoc.



Identifying Application Behavior with Visual Trace: Creating New Views

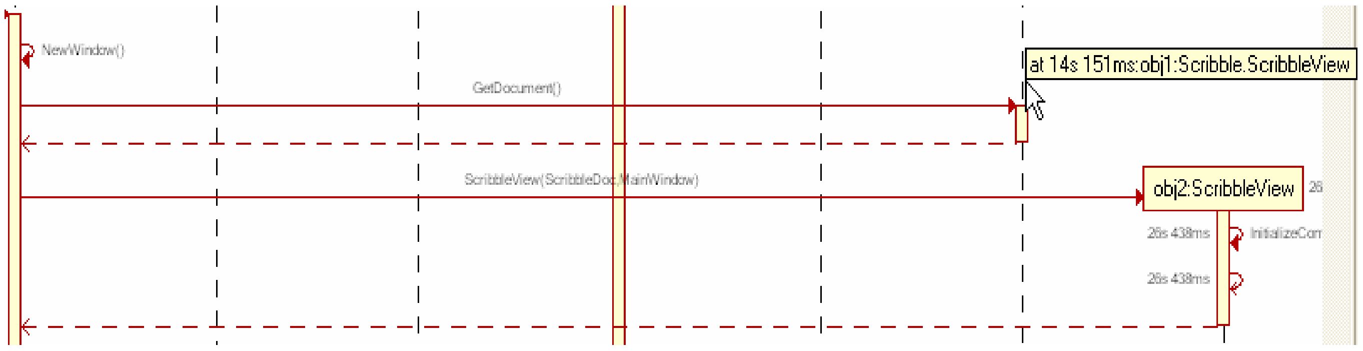
40. Scroll further down in your trace diagram to where obj0:Scribble.MainWindow is notified that a NewWindow menu event has occurred.


MenuItemHandler(Object,E...
at 5s 508ms:obj0:Scribble.MainWindow
26s 438ms
NewWindow()
26s 438ms
GetDocument()

41. Scroll to the right and you will see obj0.Scribble.MainWindow obtaining the current ScribbleDoc document from the active view, obj1:ScribbleView. Next, obj0.MainWindow creates a new ScribbleView object, ojb2.ScribbleView, associated with the current document.
42. Scroll to right and locate the obj2.System.DrawingPen lifeline. (Remember, you can identify the correct lifeline by holding the mouse button down over the line until the name appears.)
43. Right-click on the lifeline and click **Filter Lifeline** to remove this lifeline from the view.
44. Right-click in the trace diagram and click **Zoom Out** to make the diagram more readable.

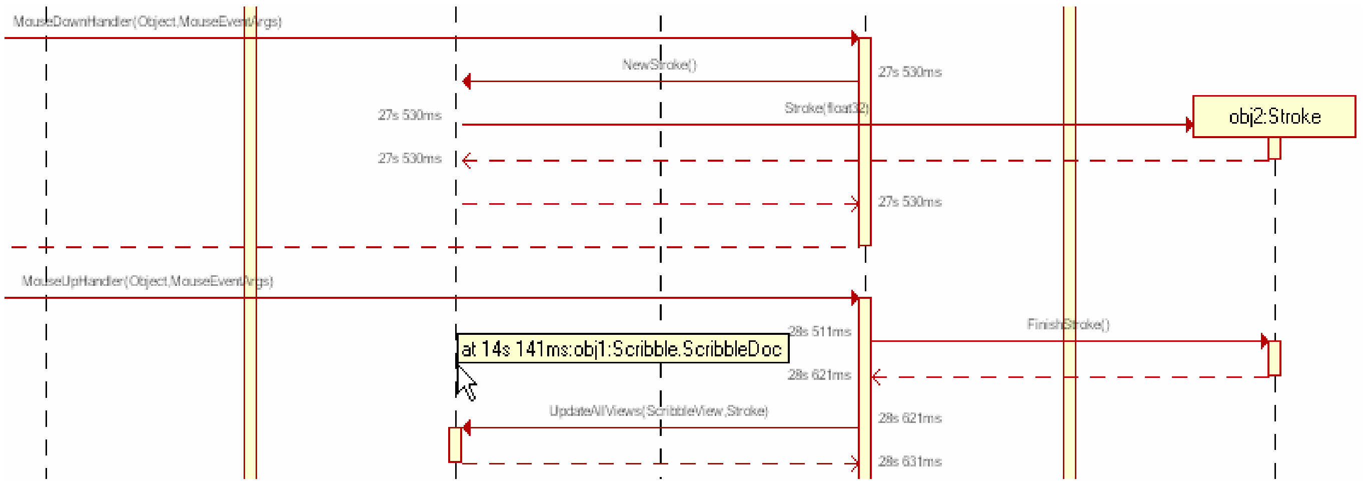
Key Benefit

Trace diagrams come with a number of formatting options so that you can visually identify key application behavior quickly.


NewWindow()
GetDocument()
at 14s 151ms:obj1:Scribble.ScribbleView
ScribbleView(ScribbleDoc,MainWindow)
obj2:ScribbleView
26s 438ms
InitializeCom
26s 438ms

45. Scroll down and locate the MouseDownHandler(Object,MouseEventArgs) message. This message occurred when you held the mouse button down to draw in the ScribbleDoc2:1 window. It's worthwhile to understand how both views are updated when you modified the ScribbleDoc document with which they were associated. Below you can see that the active view handles the mouse input. When the mouse button is released, this indicates that the drawing process is complete. The active view then notifies the active document, which then updates every view associated with it. From this

you can deduce that a view must maintain the document with which it is associated, and a document must maintain a list of all the views associated with it.


MouseDownHandler(Object,MouseEventArgs)
NewStroke()
27s 530ms
Stroke(float32)
obj2:Stroke
27s 530ms
27s 530ms
27s 530ms
MouseUpHandler(Object,MouseEventArgs)
28s 511ms
FinishStroke()
at 14s 141ms:obj1:Scribble.ScribbleDoc
28s 621ms
UpdateAllViews(ScribbleView,Stroke)
28s 621ms
28s 631ms

46. Right-click the `UpdateAllViews(Scribbleview,Stroke)` message and click **Browse Code**. Here you can see that `ScribbleDoc` parses a list of views and determines whether they need to be updated with the recent changes to the document.

Key Benefit

Using trace diagrams, you can easily toggle between a high-level view of your application and source-level details, to gain the required level of understanding for specific application functionality.

47. Right-click in the trace diagram and click **Generate Sequence Diagram**.
48. In the **New package to contain sequence diagram** text box, rename `Diagram` to `Scribble Trace`.
49. Click **Finish**. You may need to filter additional messages or lifelines if the trace diagram is too large.
50. Close all diagrams.

Key Benefit

You can automatically document program behavior by creating standard UML sequence diagrams from your trace diagrams. Standard sequence diagrams can be used to communicate specific program behavior with all team members, to facilitate the debugging and testing process.

Benefits of *Understanding Your Code*

The Rational XDE Developer Plus Visual Trace capability saves countless hours by simplifying the process of understanding application behavior. By capturing the execution of your application in a trace diagram, the Visual Trace capability lets you visualize program behavior beyond a single stack frame. Trace diagrams also include code coverage data and timing and thread information, which help facilitate run-time analysis. You can even set breakpoints and step through your code while recording the execution of your application with Visual Trace — a powerful way to understand object interaction and program behavior.

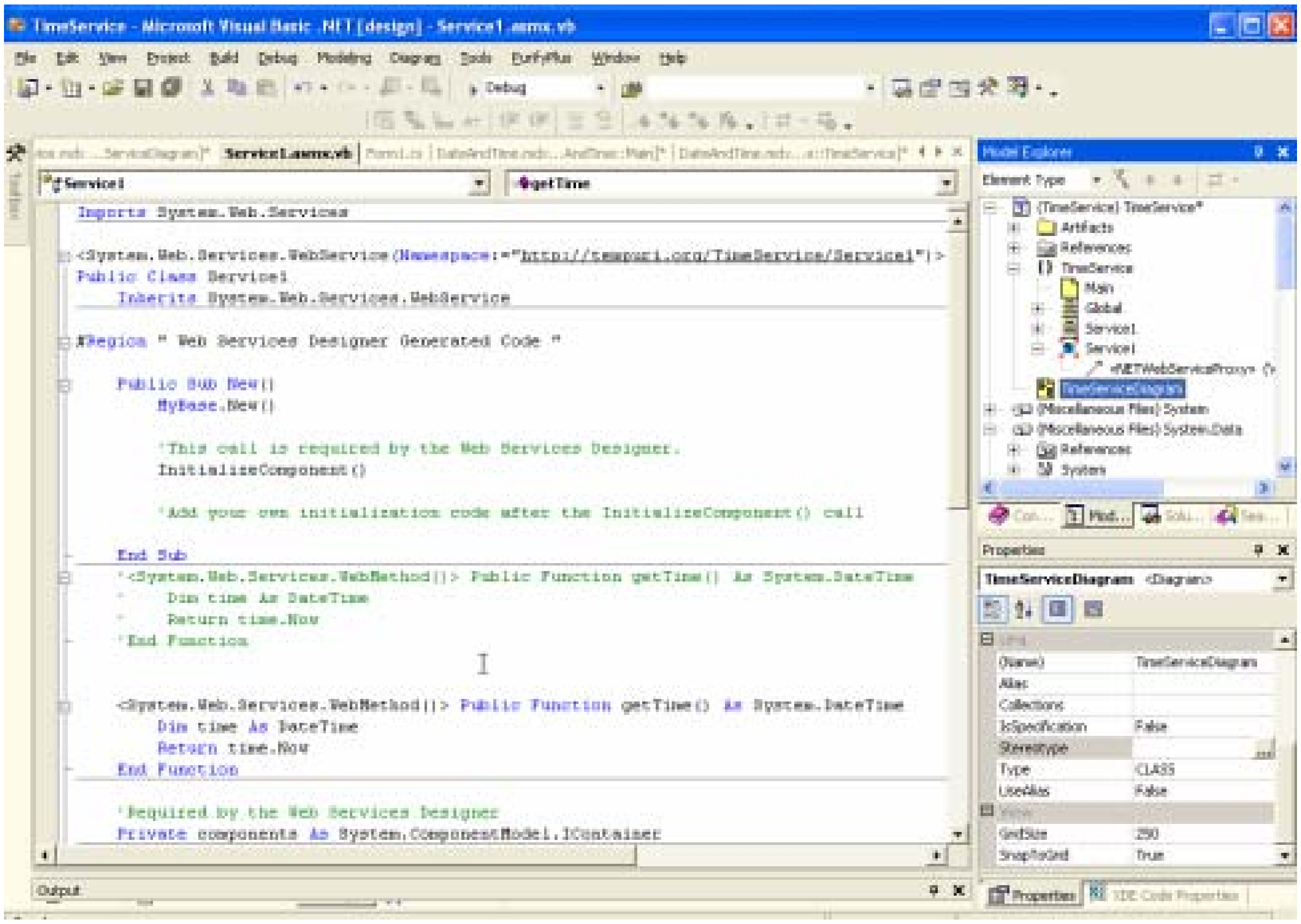
Stepping through source code is not always an effective or efficient method for understanding the behavior of your application, because this approach forces you to wade through functionality that is not essential to the basic flow of your application. Trace diagrams enable you to visualize program behavior at the appropriate level of abstraction. To identify specific program behavior faster, you can streamline the trace view by filtering out the application methods or objects that are not relevant to your current investigation. Once you have identified critical functionality in the trace diagram, you can quickly move to the source code for the details you need.

Visual Trace also provides ways to automatically document and communicate program behavior. Using trace diagrams, you can generate standard UML sequence diagrams, which can then be shared with your teams and other project stakeholders.

Developing Web Services

Rational XDE Developer's support for Web services enables you to visualize your existing systems, your new applications, and the interfaces each will require.

As a member of an IT organization, you are rarely faced with the ease of developing an isolated application. A more typical scenario is that you are concerned not only with the system you are building but also with a number of existing systems your organization has already developed, and with how the new system will fit into the overall system architecture. Using Rational XDE Developer's support for Web services, you can visualize the overall architecture and see which interfaces already exist and which ones need to be created. These can be interfaces with legacy COBOL code, Visual Basic code, or even a recently created C# application. The interfaces can be written in native code or based on industry standards such as Web services.



```
Imports System.Web.Services

<System.Web.Services.WebService(Namespace:="http://tempuri.org/TimeService/Service1")>
Public Class Service1
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent() call

    End Sub
    '<System.Web.Services.WebMethod()> Public Function getTime() As System.DateTime
    '    Dim time As DateTime
    '    Return time.Now
    'End Function


    <System.Web.Services.WebMethod()> Public Function getTime() As System.DateTime
        Dim time As DateTime
        Return time.Now
    End Function

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer
```

Figure 2. Creating Web Services — During development of Web services, Rational XDE Developer creates stereotyped UML classes to represent the Web service elements.

Experience *Developing Web Services*

In this section, you will learn how to do the following with Rational XDE Developer:

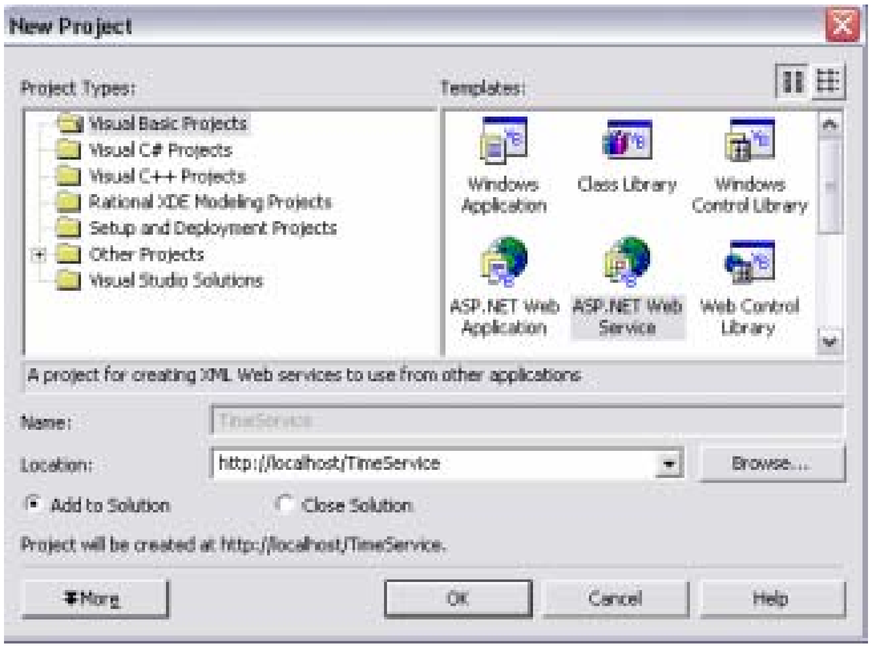
- Model Web services with ASP.NET and Visual Basic .NET
- Model key client relationships with the Web services
- Conveniently view WSDL files as UML classes

- Assisted modeling

Today, Web services are used predominantly to interface with legacy systems developed with various technologies, including Visual Basic and COBOL. Web services provide a standard protocol using SOAP, WSDL, and UDDI to interface between the same or different technologies rather than create language-specific interfaces. In this exercise, you will create a Web service that retrieves the current time. You will then update a legacy application, the `DateAndTime` application, to take advantage of your new Web service.

You must have successfully completed the "Integrated Design and Development" exercise before beginning this one. Also, if you are performing this exercise for the second time, be sure to remove the existing `TimeService` Web service by deleting the `TimeService` folder in your `/Inetpub/wwwroot/` directory.

1. Turn off auto-synchronization, as follows: click **Tools > Options** and, on the left, click the **Rational XDE** folder and then the **Round-Trip Engineering** folder to see the **Auto Synchronization** options; click the **Automatic Synchronization** check box to clear it. Click **OK**.
2. With the `Test Drive` solution open, click **File > New > Project**.
3. Select **Visual Basic Projects** as the project type and **ASP.NET Web Service** as the template.
4. Change the location to `http://localhost/TimeService`.
5. Verify that the **Add to Solution** button is selected. Click **OK**.



A project for creating XML Web services to use from other applications

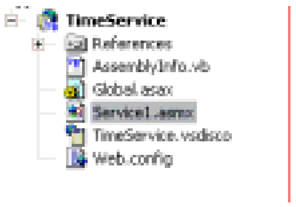
Name: TimeService

Location: http://localhost/TimeService

(● Add to Solution) ○ Close Solution

Project will be created at http://localhost/TimeService.

A Create New Web Service status message appears; once it disappears, the empty Web service has been created.

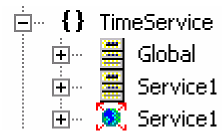


Creating a Web Service

6. In the Solution Explorer, select the `TimeService` project and click the **Synchronize** button .

7. In the Model Explorer, right-click the TimeService model ((TimeService) TimeService) and click **Add Diagram > Class**. Keep the Diagram1 default name.

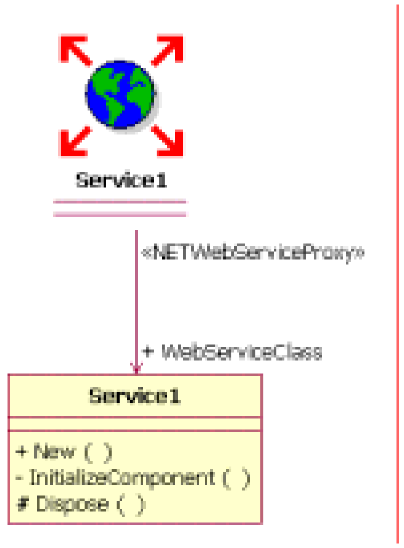
8. Expand the {} TimeService namespace to view the UML elements that were created during synchronization.



9. Drag the Service1 class and the Service1 Web service onto the class diagram. Keep the Service1 default name.

10. On the toolbar, click the **Arrange All Elements** button .

11. Select Service1 <<NETWebServiceProxy>> and click **Format > Stereotype and Visibility Styles > Shape Stereotype:Icon**.

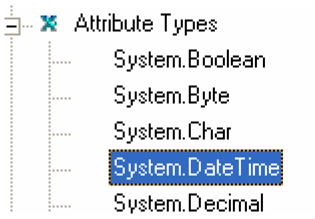


Key Benefit

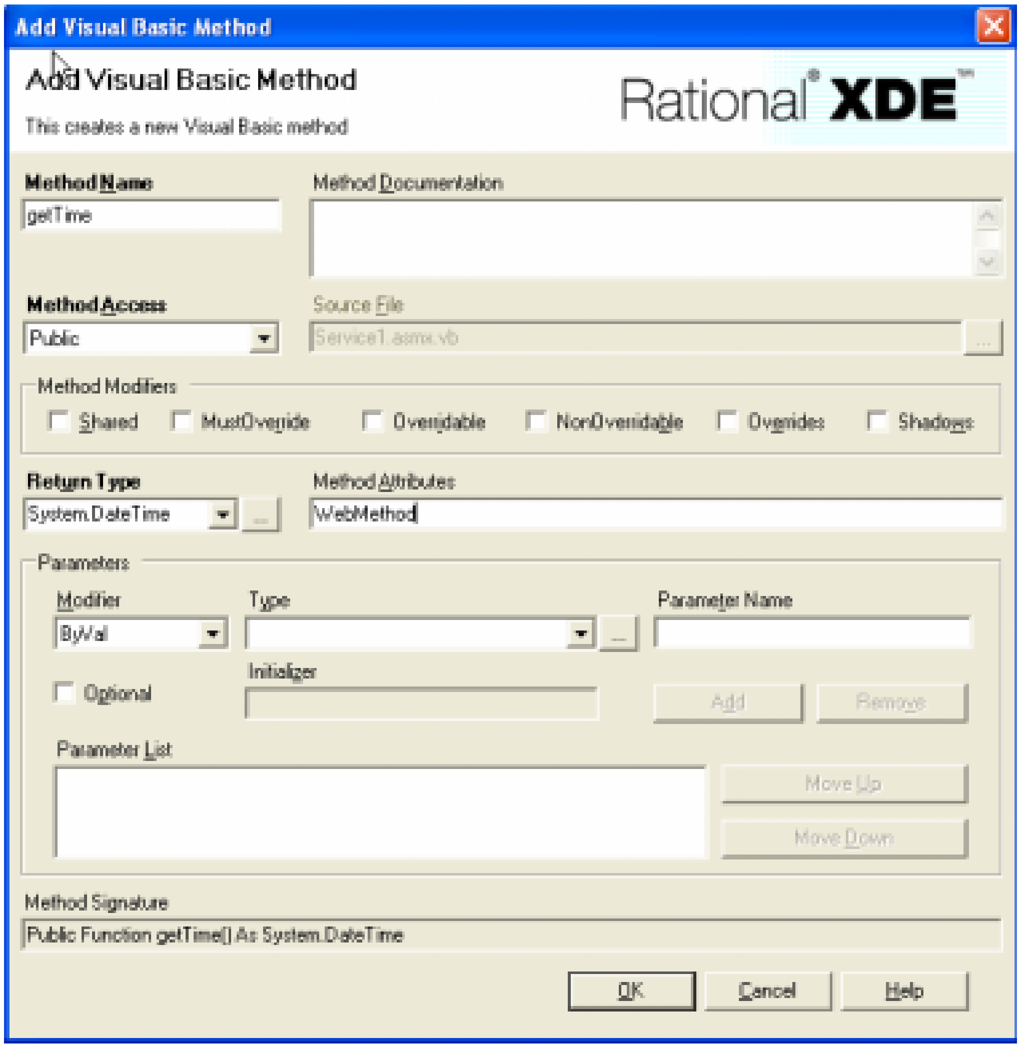
Upon synchronization of your Visual Studio Web service project, Rational XDE Developer creates two UML classes. The first class represents the XML Web service entry point, the .asmx file, and is easily recognized by the <<NETWebServiceProxy>> stereotype. The second class represents the actual implementation of the Web service, which Visual Studio .NET refers to as the **code-behind file**. Rational XDE Developer conveniently maintains a <<NETWebServiceProxy>> stereotyped relationship between the Web service class and the code-behind file. Using the UML class representations of these Web service files, you can easily model Web service functionality and even begin to create customized Web service patterns.

12. In the class diagram, right-click the Service1 class (the one with the + WebServiceClass role) and click **Add Visual Basic > Method**. Name the operation getTime.

13. In the Add Visual Basic Method dialog box, locate the ReturnType property, click the ... button, and click **Attributes Types > System.DateTime**. Click **OK**.



14. In the **Method Attributes** text box, type `WebMethod`. Click **OK**.



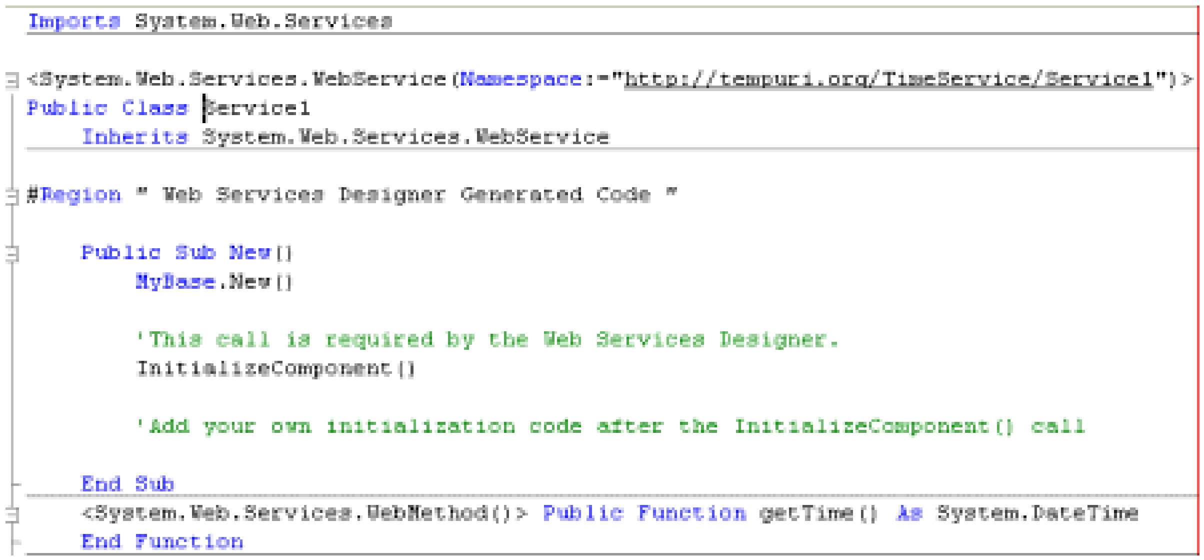
Key Benefit

Rational XDE Developer accelerates your UML learning curve by building on what you already know: C#, Visual Basic .NET, or ASP.NET. You can drag language-specific elements from the C#, Visual Basic, or Web tools in the Rational XDE Developer Toolbox to create UML models using a vocabulary you already understand.

15. In the Model Explorer, expand the `Service1` class and expand the `getTime` operation to see that it has been updated to return the `System.DateTime` type.

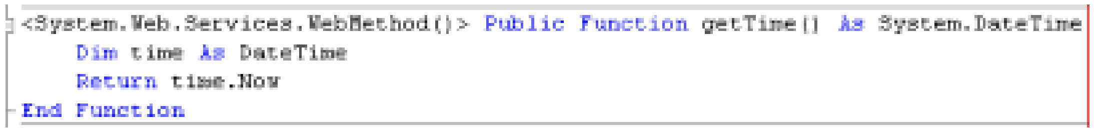
16. Select the (TimeService) TimeService model in the Solution Explorer and click the **Synchronize** button .

17. Right-click the `Service1` class in the Model Explorer and click **Browse Code** to open the `.asmx.vb` code-behind file.



```
Imports System.Web.Services

<System.Web.Services.WebService(Namespace:="http://tempuri.org/TimeService/Service1")>
Public Class Service1
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent() call

    End Sub
    <System.Web.Services.WebMethod()> Public Function getTime() As System.DateTime
    End Function
```

18. Right-click the `getTime` operation in the Model Explorer and click **Browse Code**.
19. Add the following code to the `getTime` operation:



```
<System.Web.Services.WebMethod()> Public Function getTime() As System.DateTime
    Dim time As DateTime
    Return time.Now
End Function
```

20. In the Solution Explorer, right-click the `TimeService` project and click **Save TimeService**.

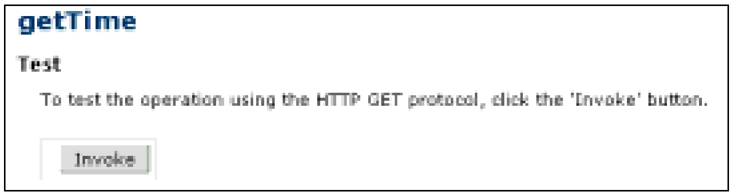
Congratulations! You have just created a Web service. Next you will test this new Web service.

Testing a Web Service

22. In the Solution Explorer, right-click the `TimeService` project and click **Debug > Start new instance**.

    Visual Studio .NET automatically creates a test harness to test your Web service.

23. In the browser window that appears, click • getTime .

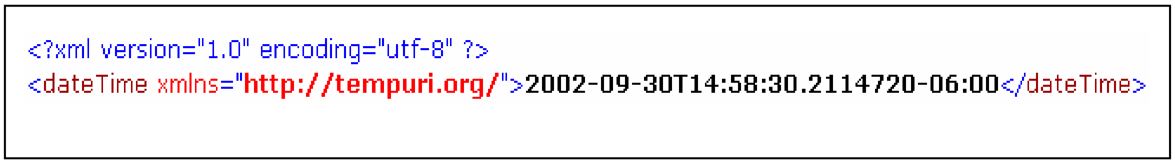


getTime

Test

To test the operation using the HTTP GET protocol, click the 'Invoke' button.

24. Click the **Invoke** button. The following display (showing the current date and time) indicates that your Web service has been successfully created.



```
<?xml version="1.0" encoding="utf-8" ?>
<dateTime xmlns="http://tempuri.org/">2002-09-30T14:58:30.2114720-06:00</dateTime>
```

25. Close both browser windows that were opened by this exercise.

Using a Web Service in Your Legacy Applications

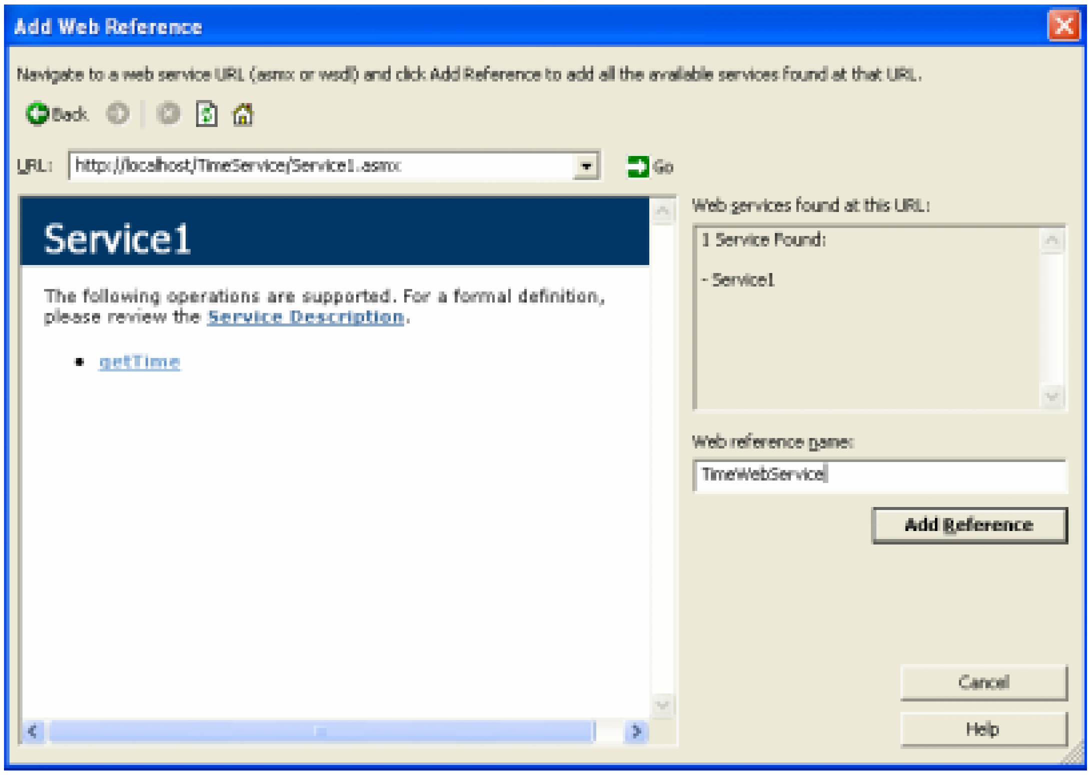
Previously you created a C# application, `DateAndTime`, to get the current date and time. You will now update this existing application to obtain the current date and time using your new Web service.

26. In the Solution Explorer, right-click the `DateAndTime` project and click **Add Web Reference**.

Key Benefit

Reviewing a WSDL XML document to determine Web service functionality can be a tedious task. Rational XDE Developer conveniently displays the WSDL XML document as a UML class so that you can quickly identify and understand the functionality available from the Web service. When you add a Web reference in Visual Studio .NET, Rational XDE Developer creates two new UML classes: the first represents the WSDL document and the second represents the proxy class generated by Visual Studio .NET. Using the UML representation of the proxy class, you can model client interactions with the Web service as if it were a locally available component.

27. In the **URL** text box, enter `http://localhost/TimeService/Service1.asmx`. Click **Go** to the right of this box.



Navigate to a web service URL (asmx or wsdl) and click Add Reference to add all the available services found at that URL.

URL: http://localhost/TimeService/Service1.asmx

Service1

The following operations are supported. For a formal definition, please review the Service Description.

- getTime

Web services found at this URL:

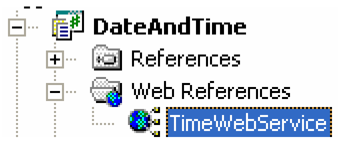
1 Service Found:

- Service1

Web reference name: TimeWebService

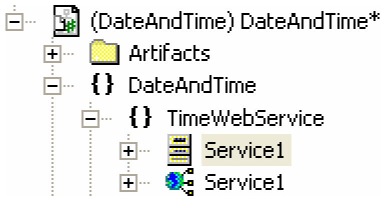
28. Enter `TimeWebService` for the Web reference name.

29. Click the **Add Reference** button. In the Solution Explorer, notice that the DateAndTime project now includes a Web reference to the TimeService Web service.



30. Select the DateAndTime project in the Solution Explorer and click the **Synchronize** button .

In the Model Explorer, you can see that during synchronization Rational XDE Developer created the {} TimeWebService namespace in the DateAndTime model.

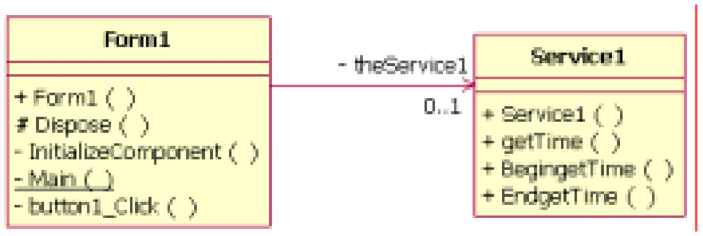


31. In the Model Explorer, right-click the DateAndTime model file ((DateAndTime) DateAndTime*) and click **Add Diagram > Class**. Name the diagram TimeService.

32. Expand the {} DateAndTime namespace and drag the Form1 class () onto the diagram.

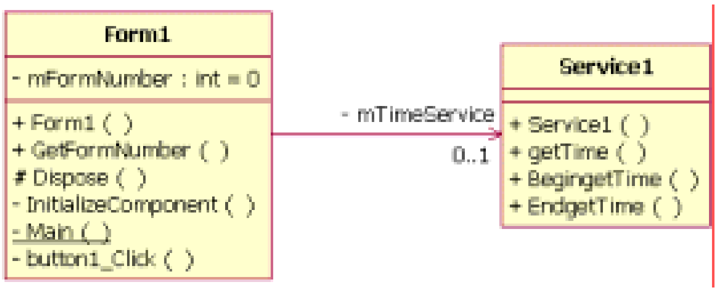
33. Expand the {} TimeWebService namespace and drag the Service1 class () onto the diagram.

34. Click the **Directed Association** tool in the Toolbox and drag from Form1 to Service1.



35. While the association is still highlighted, go to the Code Properties window and set the values indicated in the table below. (If the Code Properties window is not currently open, you can open it by clicking **View > Other Windows > Code Properties**.)

Name	mTimeService	Rational XDE Developer will generate a new member variable in Form1, mTimeService of type Service1.
Initial Value	new DateAndTime.TimeWebService.Service1()	You can set the initial value of the variable. In this case you will use a fully qualified name to assign mTimeService to a new instance of the Service1 class, which implements your TimeService Web service.



36. In the Model Explorer, right-click on the DateAndTime model and select **Synchronize**.
37. Right-click Form1 and click **Browse Code**. Notice the new member, mTimeService (whose initial value is exactly what you specified in step 35):

    private TimeWebService.Service1 mTimeService = new DateAndTime.TimeWebService.Service1();
38. In the Model Explorer, right-click the Form1 button1_Click operation and click **Browse Code**.
39. Replace the function body to invoke the Web service call instead:

```
private void button1_Click(object sender, System.EventArgs e)
{
      textBox1.Text = mTimeService.getTime().ToString();
}
```

40. In the Solution Explorer, right-click the DateAndTime project and click **Debug > Start new instance**. If you run into a namespace compilation error, verify that you entered the initial value properly in step 35, as new DateAndTime.TimeWebService.Service1().
41. Click the DateAndTime application button to see your Web service in action; then quit the application.
42. Close the windows associated with the DateAndTime project.

Congratulations! You just integrated a Web service into your existing application using Rational XDE Developer.

Benefits of *Developing Web Services*

Rational XDE Developer speeds up your learning and adoption of Web services by letting you reverse-engineer a WSDL XML document into a UML class, helping you quickly identify and understand the Web service functionality. When you create Web services in Visual Studio .NET, Rational XDE Developer creates UML classes to represent the .asmx file and the "code-behind" implementation file and maintains a stereotyped relationship between them. Using these UML classes, you can model Web services as you would any other component of your application; you can even incorporate them into patterns, to implement corporate standards.

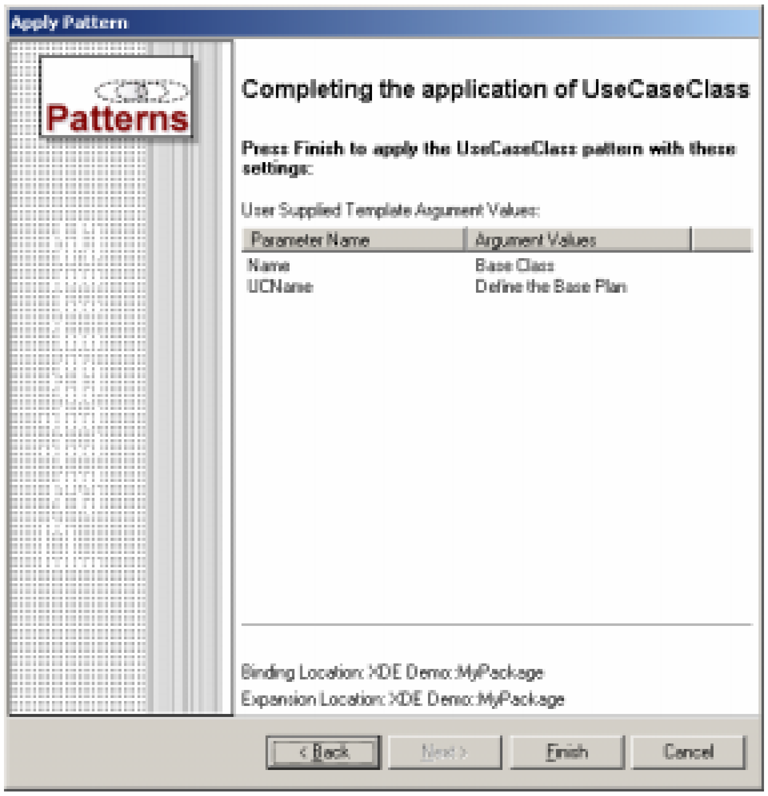
Integrating with legacy applications, learning new technologies such as Visual Basic .NET, and adopting industry standards such as UML and Web services are just some of the challenges you face daily that can be facilitated by Rational XDE Developer. Language-specific assisted modeling (for both C# and Visual Basic .NET) lets you use your knowledge of your favorite language to model. Additionally, the Code Properties window, the Properties window, and code generation combine to help you learn UML as you code, without any downtime. With Rational XDE Developer, you can model your system and all of its components, including the creation and consumption of Web services, so that you can meet the biggest challenge IT organizations face today: integrating disparate legacy systems.

Development with Custom Patterns and Templates

Reusing development assets is critical in jump-starting any software development effort. Within Rational XDE Developer, template models, a complete pattern engine, and code templates accelerate early development, help ensure efficiency and quality throughout the development lifecycle, and encourage code standardization, for more consistent, reliable projects.

You can begin by using a template model to provide structure for your project. You can apply well-known patterns, such as the Gang of Four (GoF) patterns, or for true power develop and reuse your own patterns. Patterns can easily be developed using Rational XDE Developer's built-in pattern engine and pattern wizard, enabling you to take advantage of existing code or models. All of these patterns can be shared and reused by your team members.

Code templates can be imported directly into your models and included as part of your patterns. This eliminates the hassle of writing the same code over and over again, and improves the quality of your software by enabling you to reuse proven code.



Completing the application of UseCaseClass

Press Finish to apply the UseCaseClass pattern with these settings:

User Supplied Template Argument Values:

Parameter Name	Argument Values
Name	Base Class
UCName	Define the Base Plan

Binding Location: XDE Demo::MyPackage
Expansion Location: XDE Demo::MyPackage

Figure 3. Patterns — Create and apply patterns to maximize your efficiency and effectiveness and improve your software quality.

Experience *Development with Custom Patterns and Templates*

In this section, you will design a computerized version of a board game and gain experience with the following key Rational XDE Developer features:

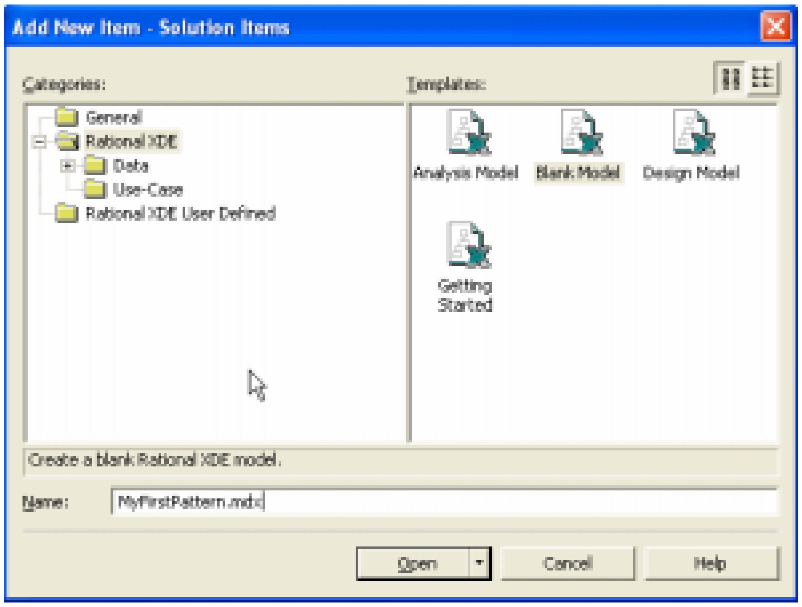
- User-defined patterns
- Code templates

The board game has a single instance of a `GameBoard` class. When you need to make sure an application has only one instance of a given class, that instance is called a **singleton**. The Gang of Four Singleton pattern ensures that a class has only one instance, and provides a global point of access to it. The `Singleton` participant defines an `Instance` operation that lets clients access its unique instance.

In this exercise, you will create your own version of the Gang of Four Singleton pattern. You will then create a custom code template and bind it to your pattern to generate code each time your pattern is applied.

You will start by creating a new Rational XDE Developer model in which to store your pattern. We recommend storing pattern specifications in models that are separate from your main development models so that the patterns can be reused in different projects and solutions.

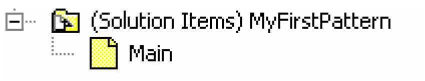
1. Turn on auto-synchronization, as follows: click **Tools > Options** and, on the left, expand the **Rational XDE** folder and then the **Round-Trip Engineering** folder to see the **Auto Synchronization** options; select the **Automatic Synchronization** check box. Click **OK**.
2. In the Solution Explorer, right-click the `Test Drive` solution and click **Add > Add New Item**.
3. Select **Rational XDE** as the category and **Blank Model** as the template.
4. Name the model `MyFirstPattern.mdx`. Click **Open**.



Create a blank Rational XDE model.

Name: MyFirstPattern.mdx

The Model Explorer displays the `MyFirstPattern` model with its `Main` diagram.

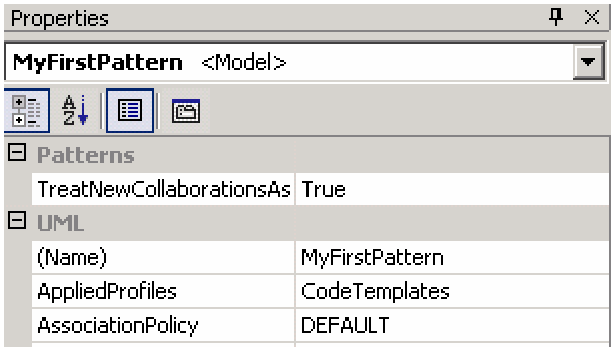


Key Benefit

Rational XDE Developer automatically adds a project to the solution when you create a new model file in that solution. Rational XDE Developer models can be shared between Visual Studio .NET projects and solutions. This capability, along with Rational XDE Developer's multiple-model support, facilitates the creation of enterprise solutions by providing an easy way for everyone to build on the same system architecture, and a way to cleanly partition the architecture and manage system/development complexity.

5. In the Model Explorer, right-click the `(Solution Items) MyFirstPattern` model and click **Properties Window**.

6. In the Properties window, select the AppliedProfiles property and type CodeTemplates in the value column. Be sure to type it as one word and respect the capitalization. Click elsewhere to make the property setting take effect.


Properties

MyFirstPattern <Model>

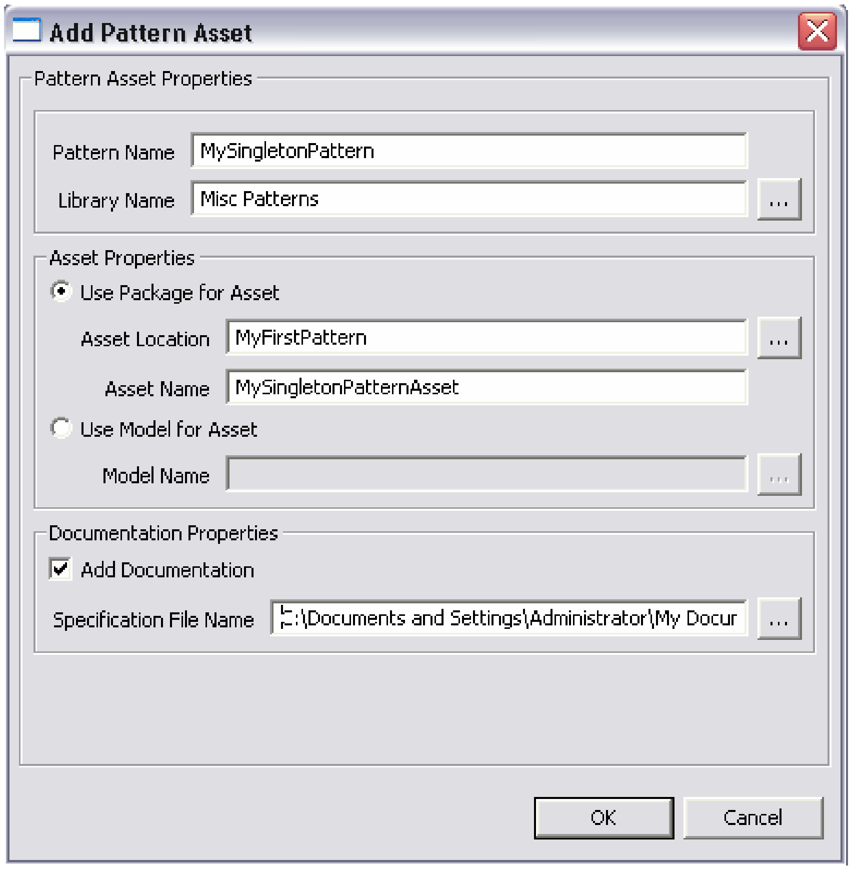
Patterns	
TreatNewCollaborationsAs	True
UML	
(Name)	MyFirstPattern
AppliedProfiles	CodeTemplates
AssociationPolicy	DEFAULT

This setting will allow you to extend the pattern with code generation capabilities to the model.

Creating a New Pattern

7. In the Model Explorer, right-click MyPatternModel and click **Add UML > Pattern Asset**.

   A **pattern asset** is a pattern that is created and packaged to be reusable.
8. In the Add Pattern Asset dialog box, set the **Pattern Name** to MySingletonPattern.
9. Under **Asset Name**, replace RASPackage1 with MySingletonPatternAsset.


Add Pattern Asset

Pattern Asset Properties

Pattern Name: MySingletonPattern

Library Name: Misc Patterns

Asset Properties

(●) Use Package for Asset

Asset Location: MyFirstPattern

Asset Name: MySingletonPatternAsset

( ) Use Model for Asset

Model Name:

Documentation Properties

[✓] Add Documentation

Specification File Name: :\Documents and Settings\Administrator\My Docur

10. Leaving the other fields at their defaults, click **OK**.

Key Benefit

Rational XDE Developer assets consist of models written and packaged to be reusable. In general, a software asset is a collection of relevant artifacts that provide a solution to a problem. In Rational XDE Developer, you can export and import Rational XDE Developer assets stored in the Reusable Asset Specification (RAS) standard format (see www.rational.com/rda), which facilitates the exchange, application, and documentation of patterns.

11. In the Model Explorer, expand MySingletonPatternAsset («Asset» MySingletonPatternAsset). Notice that MySingletonPattern is displayed with a collaboration icon ; a pattern is represented in Rational XDE Developer as a parameterized collaboration.

Key Benefit

Rational XDE Developer goes beyond providing the ability to reuse existing patterns; it lets you create your *own* patterns.

Defining Pattern Parameters

Having created a new pattern, you will now define the pattern input parameters. Rational XDE Developer records pattern parameters in **template parameters**.

12. In the Model Explorer, right-click MySingletonPattern (MySingletonPattern) and click **Add UML > Template Parameter**. Click **Yes** to reload the xml file.
13. With TemplateParameter1 selected, type InputClass to rename the template parameter. Click elsewhere to deselect the template parameter name.
14. In the Model Explorer, right-click InputClass (InputClass) and click **Add UML > Type > Class**.

    The template parameter type determines the type of input the pattern will expect, in this case a class.
15. Replace the default name with PatternClass.

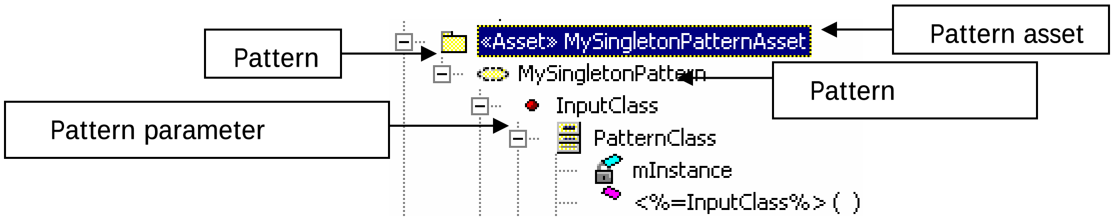
    This class will define the results of applying MySingletonPattern to classes.
16. Right-click PatternClass and click **Add UML > Attribute**. Leave the default name as Attribute1.
17. With Attribute1 (Attribute1) selected in the Model Explorer, go to the Properties window and set the following UML properties for this attribute:

Name	mInstance	Attribute name
DefaultValueExpression	null	Maps directly; be sure to use lowercase.
OwnerScope	CLASSIFIER	Attribute qualifier (maps to Java static modifier; applied to field)
TypeExpression	<%=InputClass%>	Scriptlet (see explanation below)
Visibility	PRIVATE	

When the pattern is applied, Rational XDE Developer will evaluate the scriptlet specified in the TypeExpression property, substituting the word InputClass with the name of the class specified as input to the pattern. As a result, applying the pattern to a class of type X will create a new mInstance attribute of type X.

18. In the Model Explorer, right-click PatternClass and click **Add UML > Operation**. Rename the operation as <%=InputClass%>.

    Using the same substitution mechanism, when the pattern is applied Rational XDE Developer will create a new constructor for the class to which it is applied. Later in this exercise, you will apply MySingletonPattern to a GameBoard class, which, as specified by the scriptlet, will result in a new mInstance attribute and a new GameBoard constructor for that class.



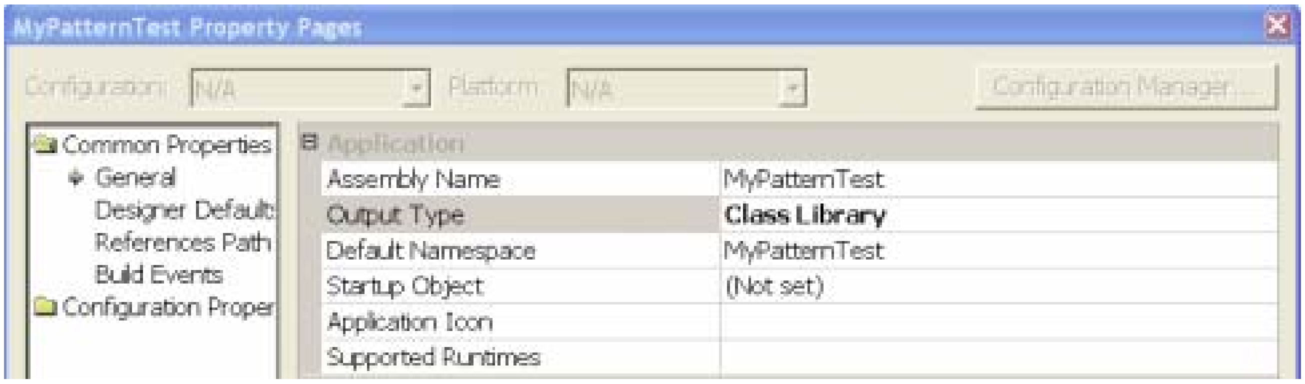
Congratulations! You have just created a new pattern.

Testing Your Pattern

Now it is time to test your pattern. You typically create a pattern as a separate, reusable artifact to be shared across projects, so you will create a separate test project to test the pattern. Since the template parameter (InputClass) for MySingletonPattern was defined with type Class, the pattern requires a class as input, so you will also create a new class.

19. In the Solution Explorer, right-click the Test Drive solution and click **Add > New Project**.
20. Select the **Visual C++ Projects** project type and the **Empty Project** template. Rename the project as MyPatternTest. Click the **Add to Solution** button. Click **OK**.

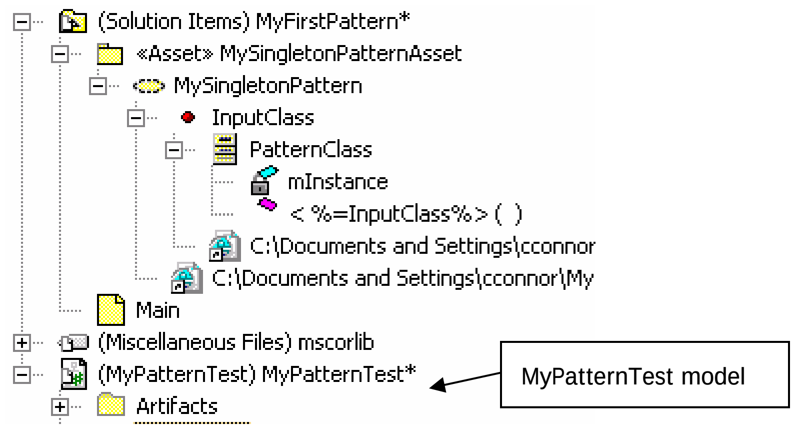
    The MyPatternTest project is added to the Test Drive solution.
21. In the Solution Explorer, right-click the MyPatternTest project and click **Properties**.
22. If not already selected, select **Configuration Properties** ➪ **General** on the left.
23. On the right, change the Configuration Type property in the **Project Defaults** section to Class Library. Click **OK**.



Application

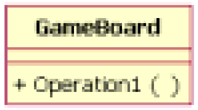
Assembly Name	MyPatternTest
Output Type	Class Library
Default Namespace	MyPatternTest
Startup Object	(Not set)
Application Icon	
Supported Runtimes	

24. With the MyPatternTest project selected in the Solution Explorer, click the **Synchronize** button ⇄ to generate a UML model for the MyPatternTest project.


(Solution Items) MyFirstPattern*
«Asset» MySingletonPatternAsset
MySingletonPattern
InputClass
PatternClass
mInstance
< %=InputClass%> ( )
C:\Documents and Settings\cconnor
C:\Documents and Settings\cconnor\My
Main
(Miscellaneous Files) mscorlib
(MyPatternTest) MyPatternTest*
Artifacts

The MyPatternTest model is displayed in the Model Explorer, and the model main diagram is automatically opened.

25. With the MyPatternTest::Main diagram active, right-click on the drawing surface and click **Add UML > Class**.
26. With Class1 selected, type GameBoard to rename the class. Click elsewhere to deselect the class.
27. Right-click the GameBoard class and click **Add UML > Operation**. Keep the defaults and click **OK**.



Applying Your Pattern

You can now apply your MySingletonPattern pattern on the GameBoard class you just created. When applying a pattern, you need to specify the pattern input parameters as well as the expansion location (that is, where you want the results of the pattern application to be stored).

28. In the Model Explorer, right-click the MySingletonPattern pattern (MySingletonPattern). Notice the **Add to Pattern Favorites** option.

    Using Add to Pattern Favorites you can add any pattern to your list of favorites to make it easily accessible.

Key Benefit

The Rational XDE Developer interface is customizable, giving you faster access to the features you use most often.

29. On the drawing surface, right-click the GameBoard class and click **Apply Favorite Pattern**.
30. Click the MySingletonPattern pattern (at the bottom of the list).

Key Benefit

The GoF patterns are available by default in Rational XDE Developer. The pattern favorites in the list can be customized via **Tools > Patterns > Organize Favorites**.

31. A dialog box is displayed, prompting you to select which style of pattern wizard you would like to use; select **Apply Pattern Wizard**. Select **Do Not Show this Dialog Again** and click **OK**.

The Apply Pattern Wizard is displayed with a short detailed description for your pattern. Since we did not take the time to enter a description when we created the pattern, you see only the boilerplate text provided by Rational XDE Developer.

32. Click **Next**.

Key Benefit

The Apply Pattern Wizard provides an easy way to reuse patterns, enabling you to enter pattern parameters via a succession of wizard screens. As you become more advanced, using the single dialog interface may prove to be a more convenient way of applying complex patterns.

MySingletonPattern requires a UML class type as input to the InputClass template parameter. Since you are running the pattern from the GameBoard class context menu, Rational XDE Developer has selected the GameBoard class for you.

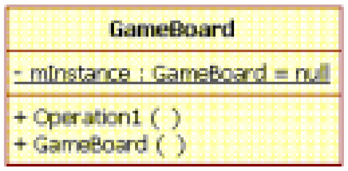
33. Click **Next**.

Key Benefit

The Apply Pattern Wizard adapts to the various patterns by displaying different screens depending on the specific input parameters for the pattern you selected to apply.

34. On the **Expansion Location** screen, click (MyPatternTest) «C#» MyPatternTe and click the **Select** button to specify your MyPatternTest project as the project in which to expand the pattern.
35. Click **Finish**, and click **OK** to dismiss the “Pattern expansion succeeded” dialog box.

The results of applying the MySingletonPattern pattern to the GameBoard class are a new mInstance static attribute (mInstance : GameBoard = null) and a new GameBoard operation. Notice that the pattern also preserved the original GameBoard class operation, Operation1.



Key Benefit

The behavior when patterns are applied is configurable such that patterns can merge, replace, or preserve existing elements. Pattern behaviors can be explored further in the Pattern Explorer and Pattern Properties windows (available from **View > Other Windows > Pattern Explorer**).

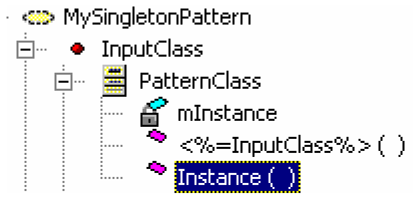
Binding Code Templates to Patterns

Now that you have validated that your pattern works as expected, you will attach some reusable code to the pattern by attaching a custom code template to the Instance operation of the MySingletonPattern pattern. Code templates can be bound to UML elements to specify the code to be generated for that UML element — in particular, method bodies.

Key Benefit

Used in conjunction with patterns, code templates provide a powerful way to get more code from your models.

36. In the Model Explorer, right-click the PatternClass class and click **Add UML > Operation**. Rename the operation as Instance.



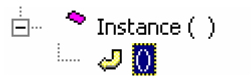
37. Right-click the Instance operation and click **Add UML > Parameter**. Keep the default Parameter1 name.

    You will use this parameter to specify the return value of the Instance method.

38. With Parameter1 selected in the Model Explorer, go to the Properties window and set the following UML properties for Parameter1:

Name	Delete Parameter1, leaving the name blank.	When the Kind property is set to RETURN, the name of the parameter is not used for code generation.
Kind	RETURN	The Instance method does not take any input parameters but needs to return one.
TypeExpression	<%=InputClass%>	The code template will substitute the word InputClass with the name of the class to which the pattern is applied.

Because the Instance method will need to return a type but that type will vary depending on the type of class to which the MySingletonPattern pattern is applied, you use a scriptlet to define the type dynamically. For example, if you apply the pattern to the GameBoard class, the GameBoard Instance operation will return a value of type GameBoard.



Key Benefit

Every time the pattern is applied to a class, that class will automatically be assigned the information stored in the pattern definition (in this exercise, an attribute and a constructor).

39. In the Model Explorer, right-click the Instance operation and click **Code Templates > Bind**.

    This will bind a code template to the pattern's Instance method.

40. In the Bind Code Template dialog box, click **New**.

    The Create Code Template Wizard appears.

41. In the **Name** box on the first of the wizard's screens, enter MyFirstCodeTemplate.
42. Optionally enter some descriptive text for the code template in the **Description** box. Click **Next**.
43. On the **Step 2 of 2** screen, click **Add**.

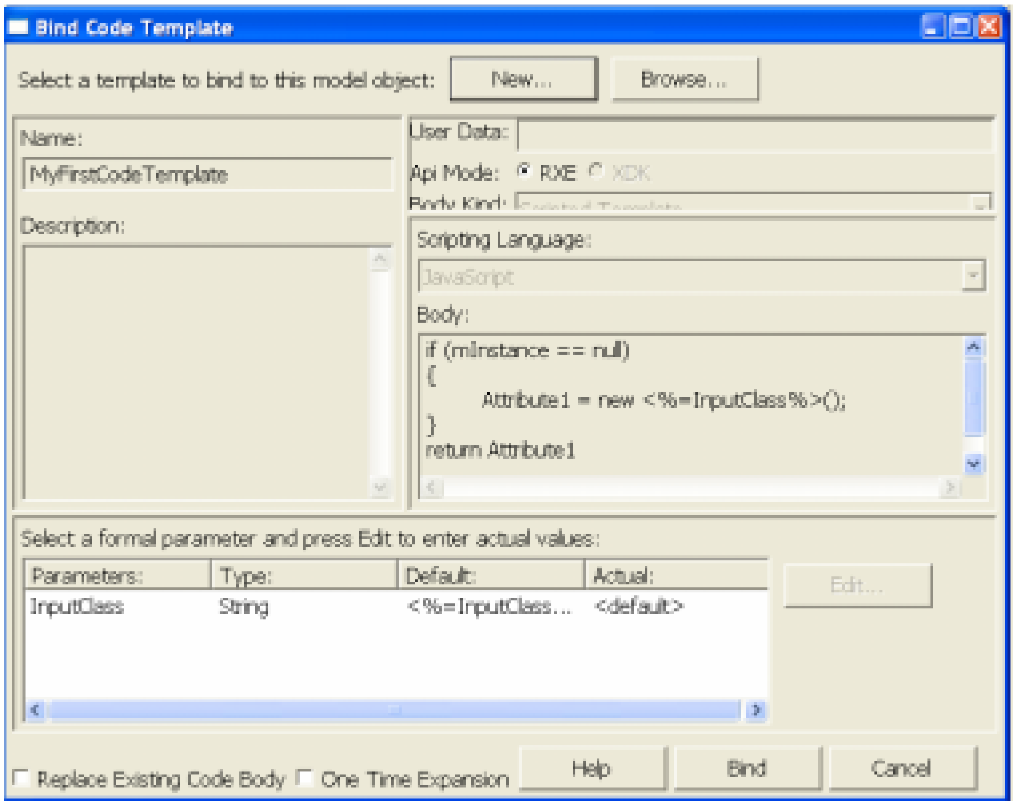
44. Fill in the Template Parameter dialog box as follows:

Name	InputClass	The InputClass template parameter will be used by the code template to get information.
Type	String	The value extracted from the InputClass template parameter will be of type String.
Default	<%=InputClass%>	The default value will be the result of the <%=InputClass%> scriptlet evaluation, which in this case is the name of the input class when the pattern is applied.

45. Click **OK**.
46. In the **Body** text box, enter the following code:

```
if (mInstance == null)
{
      mInstance = new <%=InputClass%>();
}
return mInstance;
```

47. Click **Finish**.



Name: MyFirstCodeTemplate
Description:
User Data:
Api Mode: (● RXE ○ XDK)
Body Kind: Scripted Template
Scripting Language: JavaScript
Body:

```
if (mInstance == null)
{
      Attribute1 = new <%=InputClass%>();
}
return Attribute1
```

Select a formal parameter and press Edit to enter actual values:

Parameters:	Type:	Default:	Actual:
InputClass	String	<%=InputClass...	<default>

☐ Replace Existing Code Body ☐ One Time Expansion

48. Click **Bind** to bind the code template to your MySingletonPattern pattern.

Congratulations! You just created a code template and bound it to a pattern. Now every time you apply the MySingletonPattern pattern to a class and generate code, Rational XDE Developer will generate this code for you.

Key Benefit

Customizable patterns and code templates are a powerful option for code generation. You can save time by using proven patterns and automating mundane coding through code templates. These assets make it easier for you to share your expertise and custom solutions with other members of your team. Additionally, pattern bindings make it simple to update classes when a pattern changes.

Generating Code from Code Templates

To see the result of binding a code template to your pattern, you will reapply MySingletonPattern to the GameBoard class and generate code for that class.

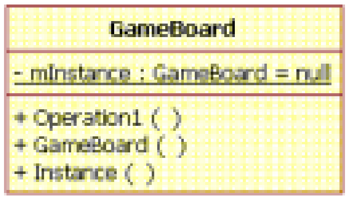
49. In the Model Explorer, scroll down to MySingletonPattern_Binding (⊞ MySingletonPattern_Binding).
50. Right-click MySingletonPattern_Binding and click **Apply This Pattern**.

Key Benefit

When you use a pattern binding, the Apply Pattern Wizard dialog boxes are populated in advance with the values from the original application of the pattern. You can walk through the dialog boxes to view your previous input or just click **Finish** to apply the pattern.

51. Click **Next**; then click **Next** again.
52. Click **Finish**, and click **OK** to dismiss the "Pattern expansion succeeded" dialog box.

    Notice the Instance operation added to the GameBoard class.

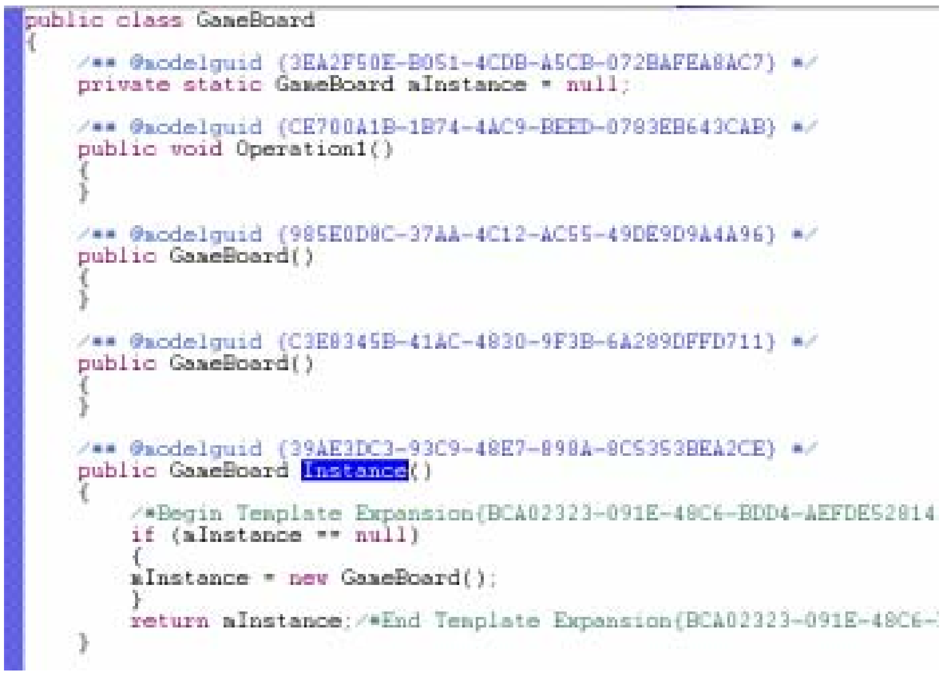
    

    GameBoard
    - mInstance : GameBoard = null
    + Operation1 ( )
    + GameBoard ( )
    + Instance ( )

53. In the Model Explorer, right-click the GameBoard class and click the **Synchronize** button.

    The synchronization process will add the Instance operation to the GameBoard class in the code and keep the code and model consistent with each other.

54. Wait until the Rational XDE Developer status bar displays "Ready" before proceeding; then, in the diagram or from the Model Explorer, right-click the `GameBoard` class and click **Browse Definition**. The code appears as follows:



```
public class GameBoard
{
    /** @modelguid {3EA2F50E-B051-4CDB-A5CB-072BAFEA8AC7} */
    private static GameBoard mInstance = null;

    /** @modelguid {CE700A1B-1B74-4AC9-BEED-0783EB643CAB} */
    public void Operation1()
    {
    }

    /** @modelguid {985E0D8C-37AA-4C12-AC55-49DE9D9A4A96} */
    public GameBoard()
    {
    }

    /** @modelguid {C3E8345B-41AC-4830-9F3B-6A289DFFD711} */
    public GameBoard()
    {
    }

    /** @modelguid {39AE3DC3-93C9-48E7-898A-8C5353BEA2CE} */
    public GameBoard Instance()
    {
        /*Begin Template Expansion{BCA02323-091E-48C6-BDD4-AEFDE52814
        if (mInstance == null)
        {
        mInstance = new GameBoard();
        }
        return mInstance;/*End Template Expansion{BCA02323-091E-48C6-
    }
```

As a result of the <%=InputClass%> scriptlet substitution in the code template body, Rational XDE Developer added the code defined in the code template to the `GameBoard Instance` method, using `GameBoard` as the return type.

55. On the main menu, click **File > Save All** to save your solution with all projects and model files.

Benefits of *Development with Custom Patterns and Templates*

Besides shipping with a set of industry-proven design patterns, Rational XDE Developer provides unprecedented support for developing and sharing your *own* patterns.

- Via a binding between the pattern and its expansions, Rational XDE Developer lets you quickly update classes to which the pattern was applied when the pattern changes.
- You can expand a pattern in any model.
- Patterns can be applied via UML stereotypes, which are a powerful way to classify model elements into categories. Once you have bound a pattern to a stereotype, applying the stereotype to the model element will automatically expand the pattern.
- Rational XDE Developer provides binding between patterns and parameterized code templates so that code is generated when you apply a pattern.
- You can easily share patterns with your team by exporting them in accordance with the Reusable Asset Specification (RAS) standard. Rational XDE Developer lets you import and export your assets to a repository that can exist locally or at an external location and be accessed via a Web service. Team members can search multiple repositories using a single keyword search and view details about the each located asset before importing it.

In short, Rational XDE Developer's complete pattern engine and code templates help accelerate early development, ensure efficiency and quality throughout the development lifecycle, and encourage code reuse, for more consistent, reliable projects. You can use your creative capabilities to design your own patterns and avoid tedious, repetitive tasks.

Flexible Team Development and Configuration Management

Rational XDE Developer works in conjunction with Rational ClearCase versions 2002.05.00 (with the latest patch) and 2003.06.00 and with Rational ClearCase LT version 2003.06.00, to provide source control, versioning, and branch/merge capabilities. You can use Rational ClearCase directly within your development environment for versioning your code or model artifacts. The result is easier, faster, more comprehensive development. Large models can be divided into multiple files, making them easier to work with and facilitating working with teams and configuration management systems. On-demand storage unit loading lets you access model elements without worrying about whether they are currently loaded; the unit simply loads automatically when you access an element that requires it.

Also, you can set your preferences to check out controlled units automatically when you edit them, or to add newly created storage units automatically to your configuration management system. Using Rational XDE Developer's compare and merge capability, you can view the differences and conflicts among model or storage unit files, manually or automatically resolve conflicts, and then merge the files to produce a single output model. This is critical in a parallel development environment, where more than one person is allowed to work on the same controlled units in parallel.

Note: Rational ClearCase and Rational ClearCase LT are separate products that are not included with Rational XDE Developer v2003.

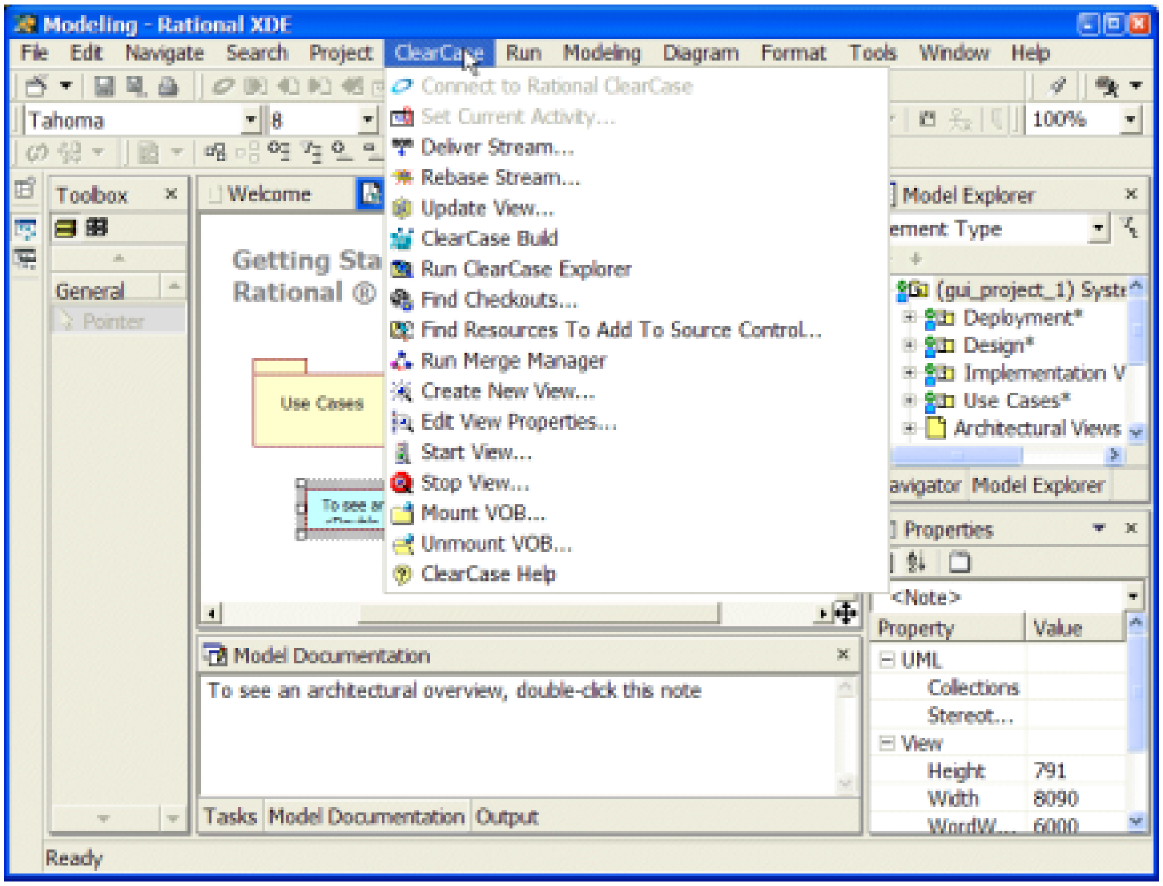


Figure 4. Rational XDE Developer and Rational ClearCase Integration — Configuration management access directly through the development environment.

Rational RequisitePro Integration

Rational XDE Developer is integrated with the requirements management tool Rational RequisitePro. This integration provides two key values to developers:

- It extends use cases in Rational XDE Developer with requirements information. This establishes a real-time window for modifying use-case attributes and traceability and viewing revision history from Rational XDE Developer. Developers can establish and maintain a bidirectional link between a use-case diagram in Rational XDE Developer and the textual definition of the use case in a RequisitePro document. The RequisitePro use-case documents contain the descriptions, flows of events, special requirements, and preconditions and post-conditions of use cases, and these documents are immediately accessible from the RequisitePro context menu that appears when you right-click a model element in the Model Explorer. In addition, use-case attributes, such as priority and status, can be set from Rational XDE Developer, enabling use cases to be sorted and filtered in RequisitePro.
- Integration with RequisitePro lets you trace requirements to design elements created in Rational XDE Developer. From Rational XDE Developer's context menus, you can add traceability links from the design element to the requirement it is implementing. This traceability information is critical to pinpointing the exact design impact of requirement changes. A change to a requirement in RequisitePro will flag in RequisitePro the affected design elements in Rational XDE Developer, by generating — automatically and in real time — suspect traceability links in RequisitePro between the changing requirement and the design element representation in RequisitePro.

Managing use cases and design artifacts in conjunction with requirements means that the scope of software development projects is better managed, change is controlled, and the project's business needs can be verified. The RequisitePro integration ensures that your team is implementing the functionality agreed upon with your customers and that this functionality evolves appropriately as the business drivers change. It also enables you to visualize the impact of requirements changes to your design, so that you know exactly which part of the design to update when requirements change.

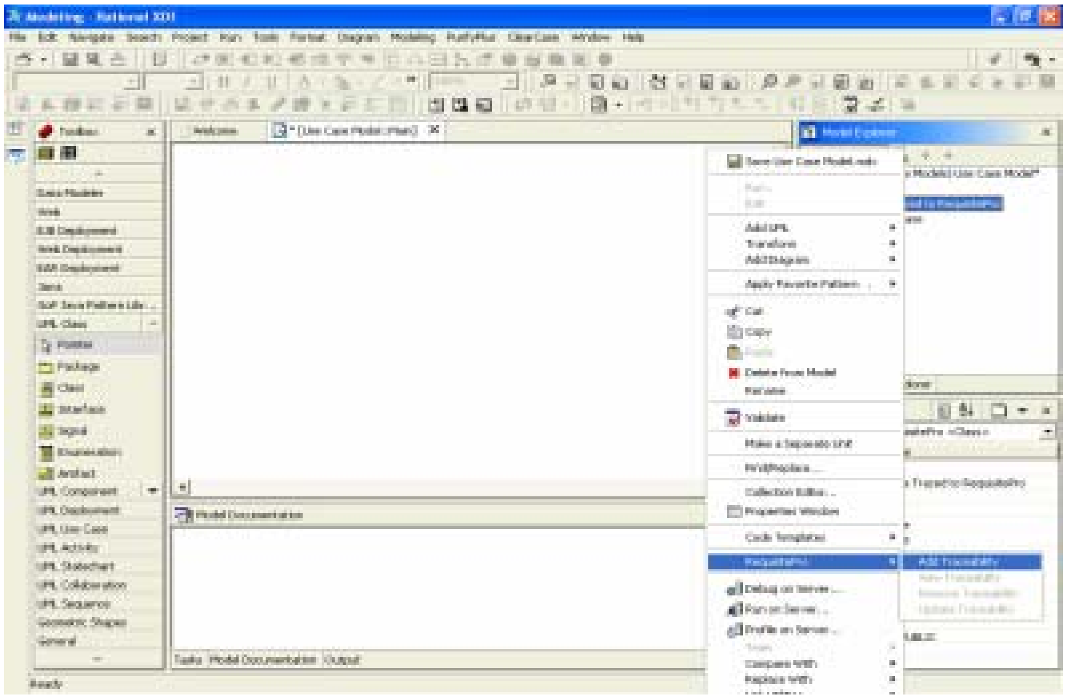


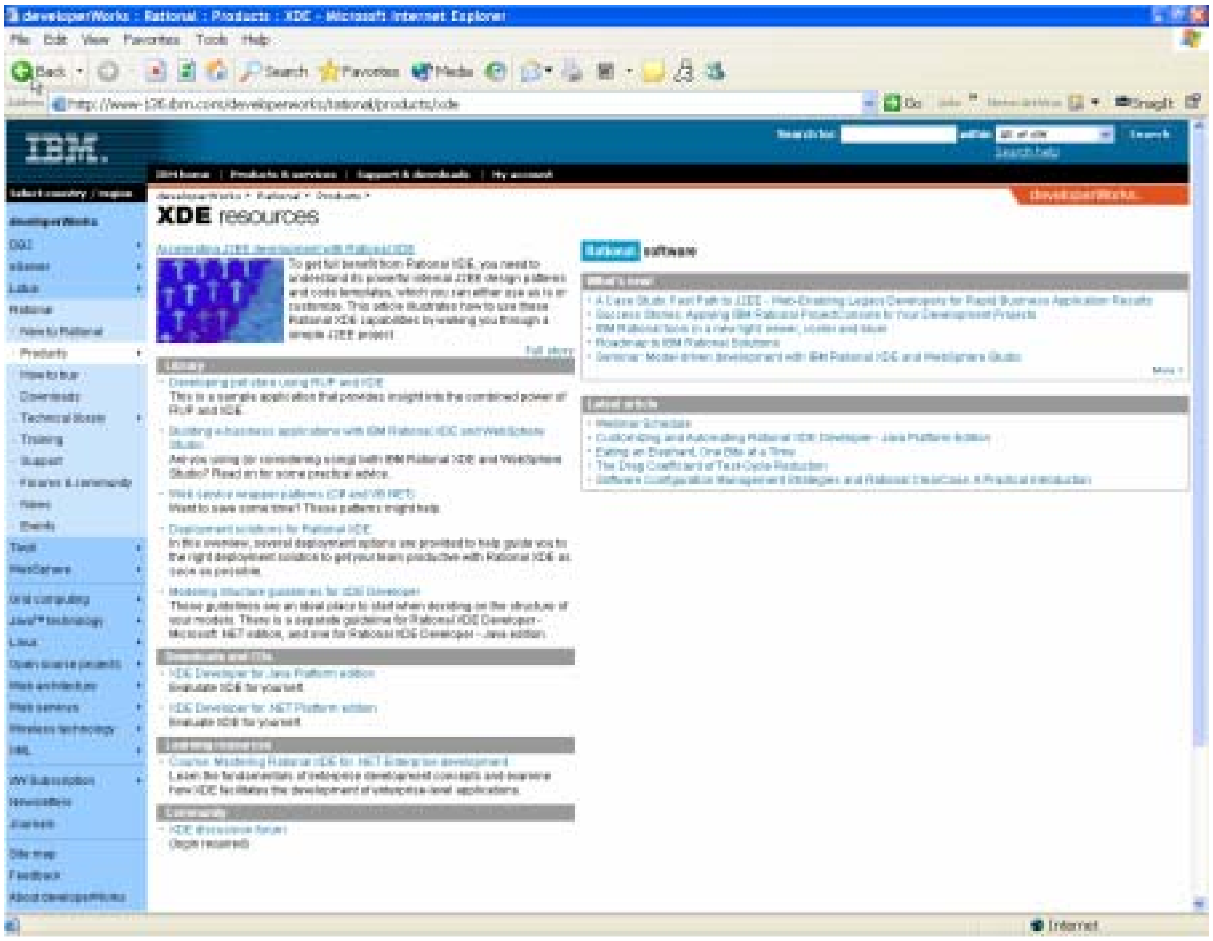
Figure 5. Rational XDE Developer and Rational RequisitePro Integration — Trace design elements to requirements by integrating with RequisitePro.

Extending Your Development Experience

Beyond providing valuable development tools that are integrated right into your IDE, IBM provides powerful resources for Rational XDE Developer practitioners through IBM developerWorks. The Rational deomain on developerWorks Web site can be found at http://www.ibm.com/developerworks/rational aggregates all the essential related content and provides access to the Rational XDE Developer community, with specific content and skill-building resources to help you increase your development efficiency and master Rational tools and software best practices.

Some of the items related to Rational XDE Developer that you will find on RDN are:

- XDE resources where you will find all content related to Rational XDE Developer
- Introductory technical articles written by .NET developers using Rational XDE Developer to build real-world solutions
- Technical articles and software assets (including patterns) that provide quick access to tested artifacts you can incorporate into your projects
- Developer-focused processes that provide role-specific guidance
- A public discussion forum that provides a central place for Rational XDE Developer customers to exchange ideas and experiences
- Self-paced Web-based training


developerWorks : Rational : Products : XDE - Microsoft Internet Explorer

http://www-136.ibm.com/developerworks/rational/products/xde

XDE resources

A Case Study: Fast Path to J2EE - Web-Enabling Legacy Developers for Rapid Business Application Results

Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction

XDE Developer for Java Platform edition

XDE Developer for .NET Platform edition

XDE discussion forum

Figure 6. XDE Resources on IBM developerWorks — Additional resources to further your development efforts.

In addition to the various services supporting Rational XDE Developer, software assets are available that extend the capabilities of the core Rational software solution. A software asset is a collection of relevant artifacts that provide a solution to a problem. The Patterns Exchange on developerWorks focuses on helping you manage and apply reusable assets so that you can build on your previous organizational and technical knowledge. It includes both sample and reference assets that make best practices tangible and provide a path toward systematic reuse of software assets.

Conclusion

We hope this guide has given you a good overview of what Rational XDE Developer can provide to .NET developers. Rational XDE Developer enables you to build better software faster by extending your development environment with essential, fully integrated tools. This revolutionary product from the leader in software development combines the expressiveness of visual modeling with the power of a Java development environment.

Rational XDE Developer improves the way you work by enabling you to:

- **Experience true developer convenience with a single design-to-code experience** — No longer do you need to constantly switch between applications to move back and forth between design and code. Everything is encapsulated within one environment, eliminating the need for you to be a master at arranging your application workspace on your computer.
- **Express yourself graphically and communicate effectively** — Within one tool you can draw any type of diagram and depict system architectures, designs, and more, by selecting from a large collection of shapes and figures. You also have the freedom to create your own shapes and figures, and you have on-demand UML conformance validation.
- **Organize models to meet your team's demands in a parallel development environment** — With cross-model referencing and versioning down to the class level, development teams can minimize redundant work, maintain the integrity of models, and partition solutions into manageable units so that team members can work on various pieces in parallel.
- **Model or code based on your preference and achieve both** — No longer do you need to always model first and then code, or always code and then reverse-engineer to generate a model. You have the freedom to do either, according to your preference. Additionally, as you work with either the model or the code, the integrity of both is maintained synchronously.
- **Develop higher-quality software faster by reusing patterns and code templates** — You can define your own patterns and code templates, or modify existing patterns supplied in Rational XDE Developer (as well as through the Rational Developer Network), to jump-start your development process. Reusing these assets eliminates mundane, repetitive tasks and ensures higher quality through the use of proven patterns.

If this guide has not quenched your thirst for information about Rational XDE Developer, here are additional places where you can learn more:

- The Rational XDE Developer tutorial available from Help.
- The product page for the Rational XDE product family, at Rational Rose XDE Developer - Product Overview - IBM Software.
- The Rational domain on IBM developerWorks at http://www.ibm.com/developerWorks/rational.

Appendix: Rational XDE Developer Interface Overview

This appendix provides a quick visual overview of the Rational XDE Developer interface in the Microsoft Visual Studio .NET IDE and the various user interface elements referred to in this guide.

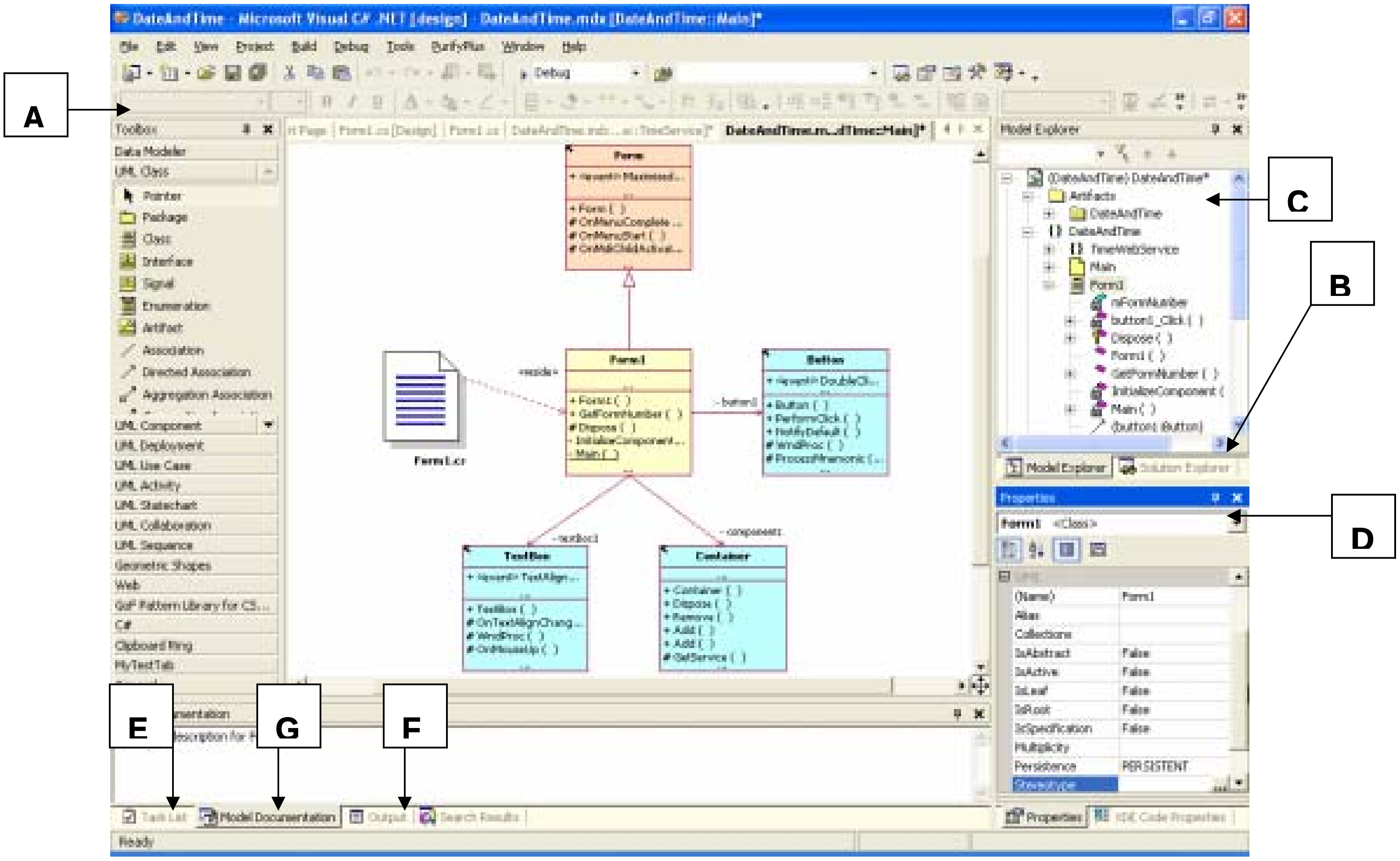


Figure 7. Rational XDE Developer Interface — Rational XDE Developer is tightly embedded in the Microsoft Visual Studio .NET IDE.

- **A.** The **Toolbox** is graphical representation of all the possible modeling elements the user can drag onto a modeling diagram. It is grouped by modeling categories such as UML diagram types and General objects used in free-form models. The up/down arrows provide navigation for lengthy lists of modeling elements. Users can add their own Toolbox categories as well as move elements between categories.
- **B.** The **Solution Explorer** provides a view of the entire project or solution. Multiple projects can be viewed at any time. A project stores a collection of files that can include references, models, source code, storage units, text files, and other project-related artifacts.
- **C.** The **Model Explorer** displays all elements that can be associated with a project, including pattern assets, diagrams, other models, and external documentation. Users can drag model elements from the Model Explorer onto a diagram. Similar to the Solution Explorer, the Model Explorer enables you to view and modify multiple models at any given time.
- **D.** The **Properties window** is context-sensitive and will display all the given properties for a selected element. These properties can be modified from within the Properties window. When you create patterns, a Properties window specific to pattern creation is opened.

E. The **Task window** shows model validation, compilation, and build errors. Users can also enter their own tasks. The tasks that appear in the window are context-sensitive: when you are viewing source code, the Task window will display compilation and build errors and warnings; when you are viewing model diagrams, model validation errors and warnings will be displayed.

F. The **Output window** is a log window that displays results of various program actions, such as the creation or modification of a new project, model file, model element, and so on.

G. The **Model Documentation window** displays documentation at the model element level.