

IBM Rational XDE Developer v2003.06.12 — Java Platform Edition

Evaluators Guide

Contents

INTRODUCTION	4
WELCOME TO RATIONAL XDE DEVELOPER	5
SUMMARY OF KEY BENEFITS	5
INSTALLING RATIONAL XDE DEVELOPER PLUS	6
J2SE DESIGN AND DEVELOPMENT	8
EXPERIENCE <i>J2SE DESIGN AND DEVELOPMENT</i>	9
Creating a New Java Modeling Project	9
Reverse-Engineering Existing Code	9
Visualizing Your Code with UML	10
Java Assisted Modeling	11
Adding a New Java Class	12
Modifying Code and the Model	13
Updating Using Multiple Selection	14
BENEFITS OF <i>J2SE DESIGN AND DEVELOPMENT</i>	15
J2EE DESIGN AND DEVELOPMENT	16
EXPERIENCE <i>J2EE DESIGN AND DEVELOPMENT</i>	16
Creating the Online Auction Project	17
EJB Modeling	17
Creating Your Web Models	18
Web Modeling with Rational XDE Developer	19
Working with Deployment Models	20
BENEFITS OF <i>J2EE DESIGN AND DEVELOPMENT</i>	21
UNDERSTANDING YOUR CODE	23
EXPERIENCE <i>UNDERSTANDING YOUR CODE</i>	23
Opening the Project and Enabling Visual Trace	23
Visually Tracing the CardApp Project	23
Manipulating Your Visual Trace Diagram	24
Identifying Application Behavior with Visual Trace	25
BENEFITS OF <i>UNDERSTANDING YOUR CODE</i>	26
DEVELOPMENT WITH CUSTOM PATTERNS AND CODE TEMPLATES	28
DEVELOPMENT WITH CUSTOM PATTERNS AND CODE TEMPLATES	28
EXPERIENCE <i>DEVELOPMENT WITH CUSTOM PATTERNS AND TEMPLATES</i>	29
Creating a Simple Project for Storing Pattern Models	30
Creating a Model for Storing Pattern Models	30
Creating a New Pattern	30
Defining Pattern Parameters	31

Testing Your Pattern.....32

Applying Your Pattern.....32

Binding Code Templates to Patterns.....33

Generating Code from Code Templates.....36

BENEFITS OF DEVELOPMENT WITH CUSTOM PATTERNS AND TEMPLATES37

FLEXIBLE TEAM DEVELOPMENT AND CONFIGURATION MANAGEMENT39

RATIONAL REQUISITEPRO INTEGRATION.....40

EXTENDING YOUR DEVELOPMENT EXPERIENCE42

CONCLUSION44

APPENDIX: RATIONAL XDE DEVELOPER INTERFACE OVERVIEW45

Introduction

Welcome to the *IBM Rational XDE Developer v2003.06.12 — Java Platform 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):
 - **J2SE Design and Development**
 - **J2EE Design and Development**
 - **Understanding Your Code**
 - **Development with Custom Patterns and Code Templates**

The exercises are independent of each other, but for the best introduction to Rational XDE Developer we recommend that you follow those of interest 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 area of DeveloperWorks, a Web site that provides a wealth of information for software professionals, including information about Rational XDE Developer, Java™ technology, and J2EE™.

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 IBM's integrated development environment (IDE) technology, Rational XDE Developer enables developers to design and code within a single environment, avoiding the need to switch between different, nonintegrated tools.

The Java Platform Edition of Rational XDE Developer includes IBM WebSphere® Studio Workbench (an IDE based on the Eclipse technology) and also integrates with IBM WebSphere Studio Application Developer (an IDE built on WebSphere Studio Workbench).

If you are already familiar with WebSphere Studio Workbench, 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 includes not only Rational XDE Developer but also IBM Rational PurifyPlus, as well as the Visual Trace capability.
- 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 Java developers (as well as one for .NET 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

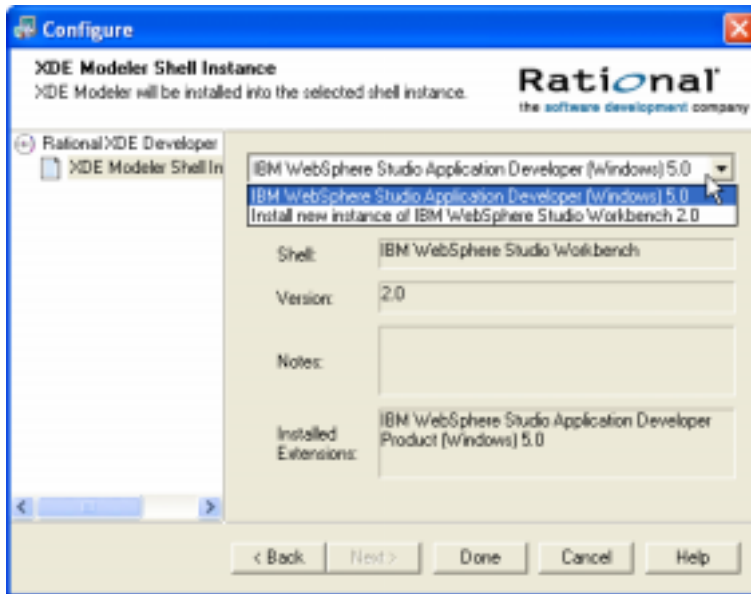
Before installing the evaluation software, first make sure your system complies with the following requirements:

Operating system	<ul style="list-style-type: none"> • Windows NT SP6a • Windows 2000 Professional, Service Pack 3 or Service Pack 4 (Service Pack 4 is recommended) • Windows 2000 Server, Service Pack 3 or Service Pack 4 (Service Pack 4 is recommended) • Windows 2000 Advanced Server, Service Pack 3 or Service Pack 4 (Service Pack 4 is recommended) • Windows XP Professional, Service Pack 1
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 the evaluation copy of Rational XDE Developer Plus:

1. If you are installing from the IBM Rational Solution for Eclipse and IBM WebSphere Developers CD:
 - a. The **IBM Rational Solution for Eclipse and IBM WebSphere Developers** page should appear; if it doesn't, double-click index.htm.
 - b. In the navigation bar on the left, click **Evaluation Registration**.
 - c. Read and accept the license agreement.
 - d. When prompted, enter the required information. You will receive an e-mail message containing your license key.
2. If you are installing from a Web download:
 - a. Unzip the Zip file you downloaded from the Web and extract its contents into a temporary directory.
 - b. Double-click index.htm.
3. In the navigation bar on the left, click **Evaluation Installation**.
4. The File Download dialog box appears; click **Open**.
5. The WinZip Self Extractor dialog box appears; click **Unzip** to unzip the installation files into the default temporary location of C:\DOCUMENT-1\current-user\LOCALS-1\Temp\.
6. Click **OK** in the message box indicating that the files have unzipped successfully. Then click **Close**.
7. Navigate to the C:\Documents and Settings1\current-user\Local Settings\Temp\DISK1\ directory and double-click Setup.exe.

8. As you're led through the installation process, click **Next** on each screen after reading it and following any instructions. At one point you will need to accept the terms of the license agreement; then you will select the folder in which to install Rational XDE Developer Plus (the default being C:\Program Files\Rational\).
9. As the last step before installation can begin, the **XDE Modeler Shell Instance** screen presents a drop-down list that lets you select the shell instance in which to install Rational XDE Developer Plus. (Rational XDE Modeler is a subset of Rational XDE Developer.) The choices are described below; change the default setting if desired and click **Done**.
 - **IBM WebSphere Studio Application Developer (Windows) 5.0 or 5.1.1** — If WebSphere Studio Application Developer is installed on your system, it will be the default choice.
 - **Install new instance of IBM WebSphere Studio Workbench 2.1.2** — If neither WebSphere Studio Application Developer nor the Eclipse shell (including any Eclipse-based product, other than a WebSphere Studio product) is detected on your system, the default choice will be to install into a new instance of WebSphere Studio Workbench (installing Rational XDE Developer as a stand-alone IDE). You can choose this option even if it is not the default for your system, although this would not be typical.
 - **Eclipse 2.0x or 2.1.2 shell or Eclipse-based product (other than a WebSphere Studio product)** You must set specific registry keys to install into the Eclipse shell. For further details on this installation option, see the Rational XDE Developer Release Notes (the readme.html file located in the folder into which you extracted the installation files in step 5).



10. Click **Install** to install Rational XDE Developer Plus in the folder indicated earlier.
11. If the **Error Summary** screen is displayed, click **Next**.
12. Copy the license key (which you received via e-mail or downloaded) to the folder in which you installed Rational XDE Developer Plus.

J2SE Design and Development

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 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 with IBM's IDE technology (WebSphere Studio Workbench or WebSphere Studio Application Developer), thereby providing a consistent look and feel and user experience.

Modeling capabilities are now as much a part of your Java IDE as your code editor, compiler, and debugger. With this combined environment, your primary development tools 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.

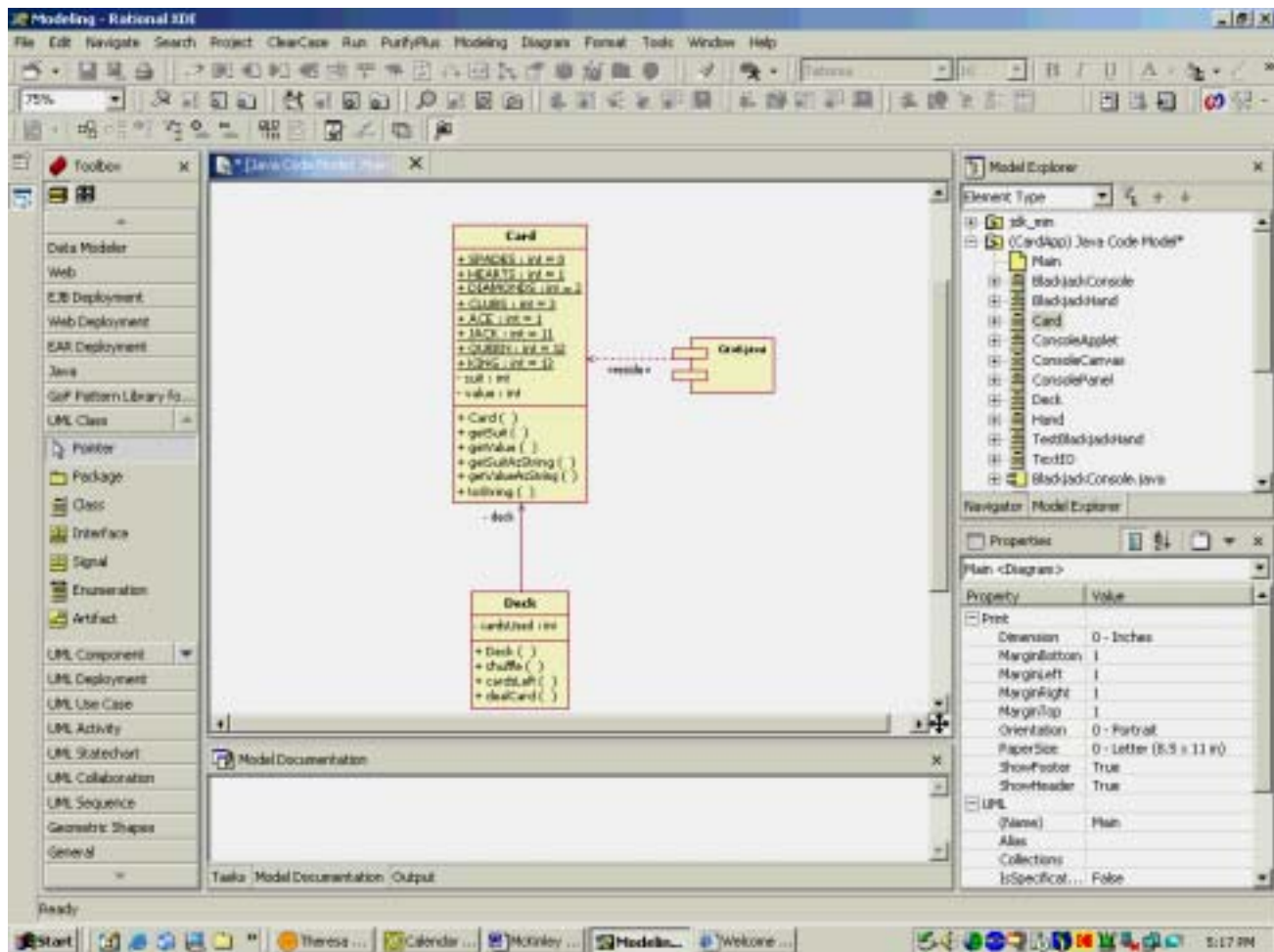




Figure 1. Configurable Synchronization — Synchronize your code and model automatically or manually.

Experience *J2SE Design and Development*

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

- Automatic or manual code synchronization
- Java Assisted Modeling
- Navigation within the Rational XDE Developer environment
- Multiple selection

Creating a New Java Modeling Project

1. Click **File > New > Project**.
2. Select **Modeling** on the left and **Java Modeling Project** on the right. Click **Next**.
3. Name the project CardApp and click **Finish**.
4. In the Model Explorer, select the Java Code Model.
5. On the main toolbar, depress the **AutoSync** button  to enable auto-synchronization. Once depressed, the button should be colored .


Note: For auto-synchronization rules, see **Window > Preferences > Rational XDE > Code-Model Synchronization**.

Key Benefit

Rational XDE Developer jump-starts your modeling efforts by providing these targeted modeling projects:

- **Basic Modeling Project** — A modeling project that does not support code generation.
- **Java Modeling Project** — A Java project that includes support for modeling Java code in UML, generating Java code, and creating patterns and code templates.
- **C++ Modeling Project** — A C++ project that includes support for modeling C++ code in UML, generating C++ code, and creating patterns and code templates. (Available in the Rational XDE Developer Plus Edition Only)
- **Web Modeling Project** — A project specifically for modeling and designing a Web application using JSP™ pages, JSP tag libraries, servlets, Enterprise JavaBeans™ (EJB™) components, and other Java elements.
- **Data Modeling Project** — A project specifically for modeling databases.
- **EJB Modeling Project (WSS AD Only)** — A project that includes an empty Java code model for EJB components and supporting Java code.
- **Enterprise Application Modeling Project (WSS AD Only)** — A project that represents the EAR file that will be deployed. It can contain up to three optional subordinate projects: an EJB modeling project, a Web modeling project, and a Java modeling project for client-side Java code.



Reverse-Engineering Existing Code

6. Click **File > Import**.
7. Select **Zip file**  **Zip file**. Click **Next**.
8. Browse to the location of your Blackjack.zip file.
9. Under **Select the destination for imported resources**, browse to select your CardApp project.

10. Click **Finish**.



Rational XDE Developer imports your source files and synchronizes the model. You can now view the UML classes that represent your Java classes.


Visualizing Your Code with UML

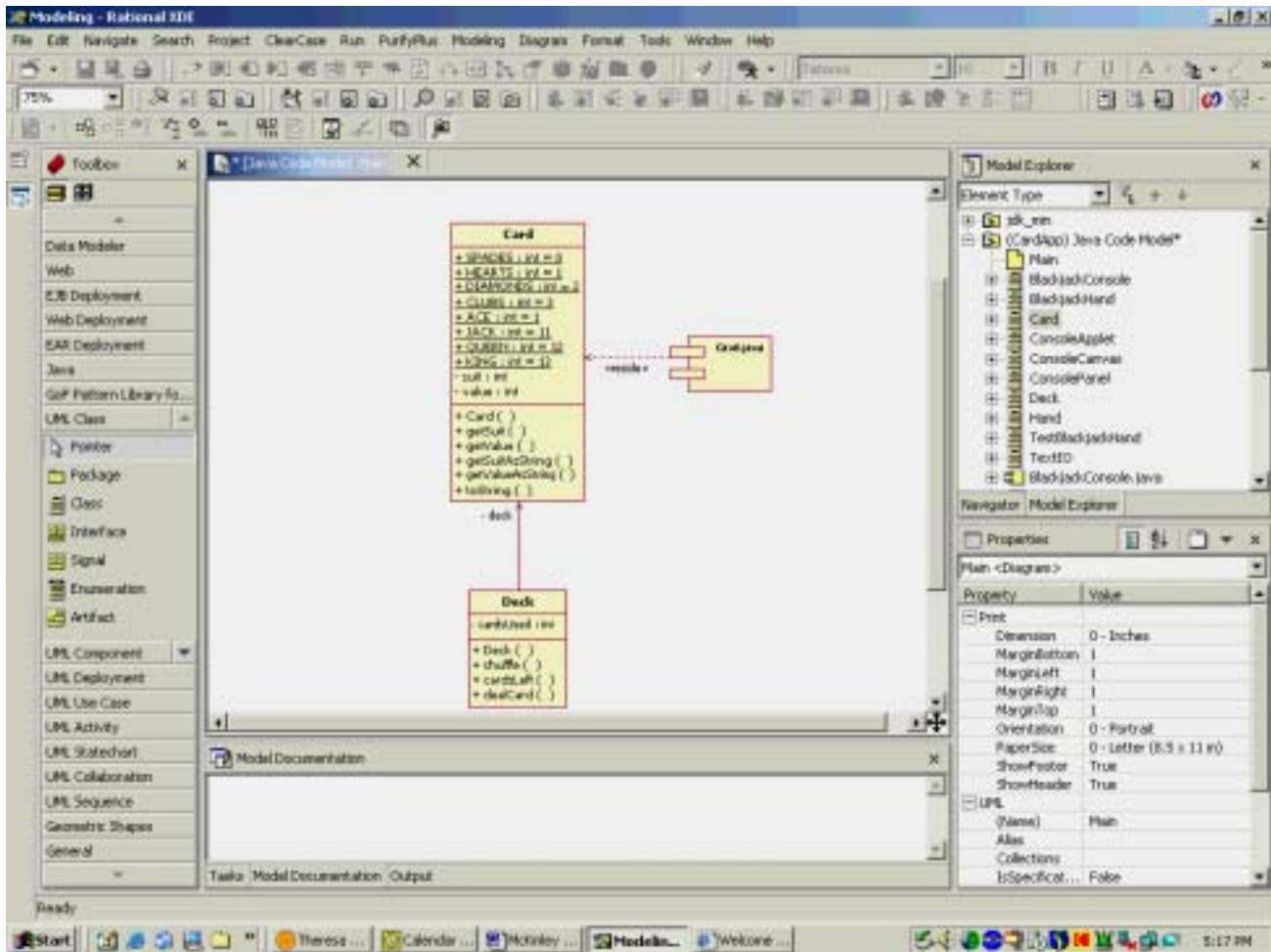
11. In the Model Explorer, drag the Card class  onto the Main diagram already opened on the drawing surface.
12. Right-click the Card class in the diagram and click **Add Related Shapes** .
13. Accept the defaults and click **OK**.

Key Benefit

You can select the models from which elements are added, as well as the direction and type of the relationships on which to base inclusion of related elements. This level of flexibility makes it easy to visualize your code at just the right level of abstraction.

14. On the main toolbar, click the **Arrange All Elements** button  to format the diagram.
Note: A second click on the button might further refine the diagram layout.
15. Also on the toolbar, change the **Zoom** factor to 75%  for better viewing.

16. Use the crosshairs  in the bottom-right corner of the diagram to navigate your diagram with a bird's-eye view. The diagram layout should look like this:

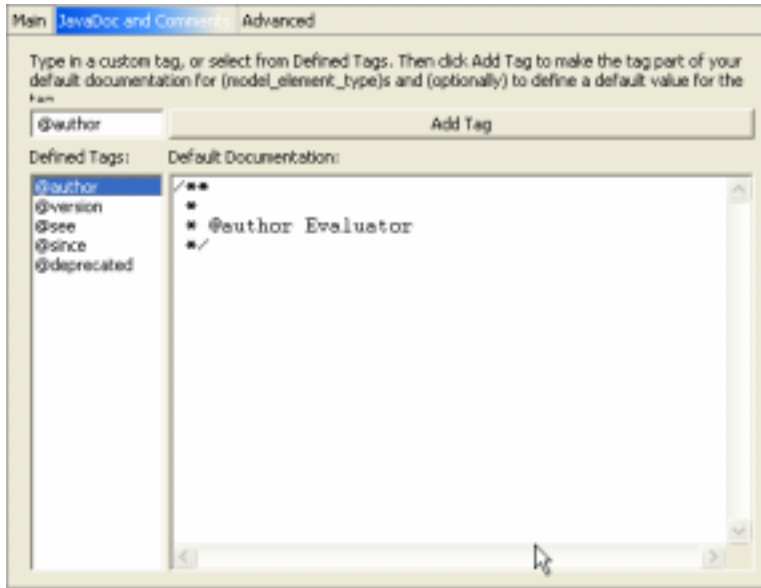


Key Benefit

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.

Java Assisted Modeling

17. On the main menu, click **Window > Preferences**.
18. Expand **Rational XDE > Java** and select **Assisted Modeling**.
19. Ensure that all options are selected. (They are selected by default.)
20. Expand **Rational XDE > Java > New Element Defaults** and select **Class Defaults**.
21. Click the **JavaDoc and Comments** tab.
22. Select the @author tag and click **Add Tag**.
23. In the editor window, type Evaluator after the @author tag.




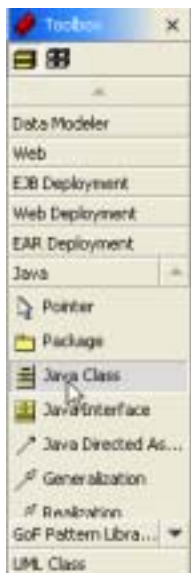
24. Click the **Advanced** tab.
25. Select the **Create a default constructor method** check box.
26. Click **OK**.

Key Benefit

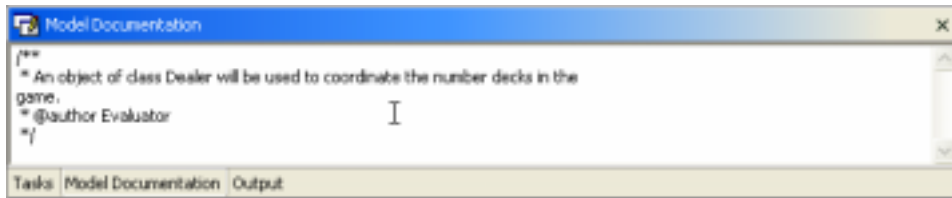
Java Assisted Modeling automatically generates a constructor and a finalizer when a class is created, as well as getters and setters for each new Java class field. Rational XDE Developer makes Java development faster and easier by automating some of the tedious, mundane tasks of development, so that you can focus on more critical and complex code.

Adding a New Java Class

27. In the Toolbox ( Toolbox , on the left), locate the tools available under the **Java** category, which lists the typical Java types used in a class diagram.



28. Drag from **Java Class** onto the drawing surface.
29. In the Create Java Class Assisted Modeling dialog box, name the class Dealer. Leave all other defaults.
30. Click **OK**.
31. If the class name appears highlighted in edit mode, click anywhere in the diagram and then click the class to reselect it.
32. Click in the Model Documentation window and type a description of the Dealer class. For example:



Key Benefit

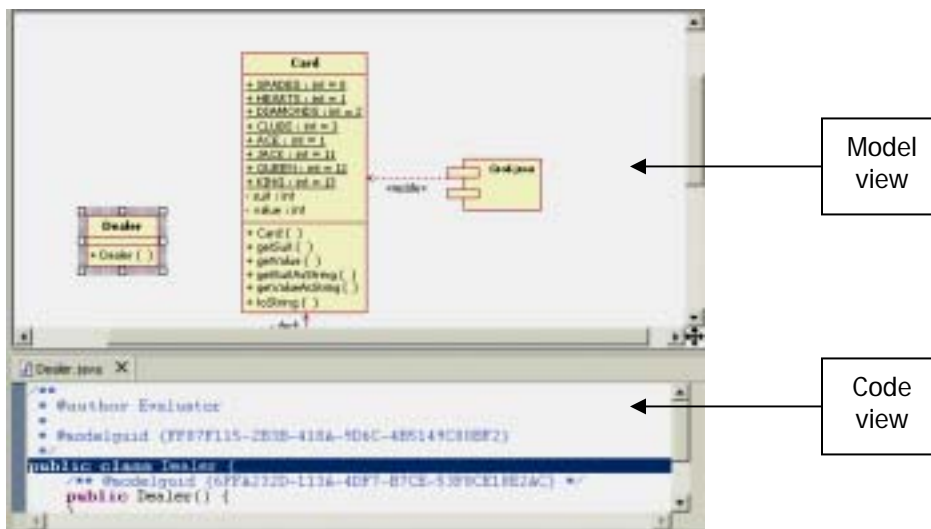
Within Rational XDE Developer, you can easily create and maintain consistent model documentation. You can jump-start documentation efforts by defining text or Javadoc defaults that are automatically applied based on user preferences. Comments created in the model are synchronized with code so that updates can be made to either location.

Modifying Code and the Model

33. Right-click the Dealer class in the diagram and click **Browse Code**.
34. In the drawing area, click the **Dealer.java** tab and drag down until the mouse pointer turns into a black arrow pointing downward; then release the mouse button.

Key Benefit

Rational XDE Developer takes advantage of the WebSphere Studio Workbench environment so that you can intuitively arrange models and code in your workspace. You can easily tile the Java-aware editor with the model view, enabling you to simultaneously view a model and the associated code. The tab-based interface provides an easy way to switch between diagrams, and it also visually indicates which diagrams are currently open.

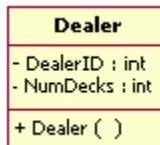


35. In the code editor, add the dealerID and numDecks attributes to the Dealer class as follows:

```
public class Dealer {
    /**
     *
     * @modelguid {8E75846F-DB28-4D4A-9D50-A19365004648}
     */
    public Dealer() {
    }
    private int dealerID;
    private int numDecks;
}
```

36. Right-click in the code editor and click **Save**.

The Dealer class is automatically updated in the model diagram, keeping the model in sync with the code.



Key Benefit

With Rational XDE Developer, you can work in the way that feels most comfortable to you, either in a model or in source code, without the hassle of having to manually keep the model and code synchronized.

Updating Using Multiple Selection

37. In the Model Explorer, locate and expand the Dealer class.
38. While holding the CTRL key down, click the two Dealer attributes to select them.
39. Right-click one of the selected attributes and click **Add Java > Getter and Setter**. Note the updates made to the Dealer class on the drawing surface and in the Dealer.java file.

Key Benefit

You don't have to waste your time adding getters and setters; Rational XDE Developer will do it for you with the click of a button.

40. In the Properties view, locate the **UML** properties group, which should already be expanded. Take a quick look at the Dealer class in the diagram before completing the next step.
41. Locate the TypeExpression property and enter long as its value. Click elsewhere to effect the property change.

The Dealer class on the drawing surface now displays each attribute as having type long, and the code has been updated. Code updates include types passed to and from the getter and setter methods.

Key Benefit

The Properties view speeds up the setting of properties by letting you select multiple elements and then modify common properties in one action. Changes made in this view are automatically reflected in

diagrams. The Properties view helps you gain an understanding of your models because it keeps element properties visible and in context as you navigate through your model.

42. In the Model Explorer, right-click the Java Code Model and click **Save Java Code Model.mdx**.

43. Close the  window.

Benefits of *J2SE Design and Development*

An integrated design and development environment increases developer productivity by providing an extended development environment in which the Java code and model are kept synchronized at all times. By handling the hassle of code-model synchronization, Rational XDE Developer enables you to work where you feel most productive, either in the model or in the code. You simply specify auto-synchronization rules to resolve any conflicts between code and model, and Rational XDE Developer handles the rest.

The IDE embedded in Rational XDE Developer enhances developer productivity with Java Assisted Modeling, which automatically generates constructors, finalizers, and getters and setters, relieving you of the burden and tedium of creating infrastructure code. In addition, the multiple-selection feature of Rational XDE enables you to apply changes to multiple elements rapidly. Used together, Java Assisted Modeling and the multiple-selection features of Rational XDE Developer offer a streamlined, automated alternative to error-prone, manual updates.

Some related Rational XDE Developer features that are not covered in this guide are also worth mentioning. You can:

- Represent inter-class dependencies as attributes or associations, and toggle freely between the two representations using simple right-click context menus
- Use Javadoc-defined and user-defined tags and values to make your model and code self-documenting
- Apply standard J2EE patterns to create EJB entity beans, session beans, and servlets

J2EE Design and Development

Team collaboration is critical to the success of any software development project, and this is especially true for complex n-tier J2EE applications. Development teams typically consist of architects, data modelers, application developers, and Web application developers, each with distinct development goals. You need to effectively communicate your design decisions regarding data, business logic, Web presentation, security, and scalability to this diverse group. Rational XDE Developer is uniquely suited for enhancing team communication by unifying the entire team with a common language, the Unified Modeling Language (UML). With Rational XDE Developer, architects, developers, and data modelers can all work together using one tool, one methodology, and one language.

Rational XDE Developer accelerates your J2EE development by providing specific models for design and deployment, adding structure to your project right out of the box. You can begin your data and application design using J2EE-specific patterns to create, for example, Container-Managed Persistence (CMP) entity beans that can then be transformed into data tables. Using the Web Modeling profile for UML, you can clearly depict relationships and information flow between Web artifacts such as HTML forms, client pages, and JSP pages (JavaServer Pages™ technology). To help ensure that your team stays abreast of any design changes, you can create diagrams that show traceability between project tiers, such as dependencies between servlets and beans. Free-form diagrams make it possible for you to communicate complex design ideas to any audience.

Rational XDE Developer facilitates all aspects of J2EE development, including deployment. Data descriptor elements such as security roles, method permissions, and transactions are represented in UML. You can model deployment scenarios once and then reuse them to speed up deployment to multiple server types or multiple server nodes.

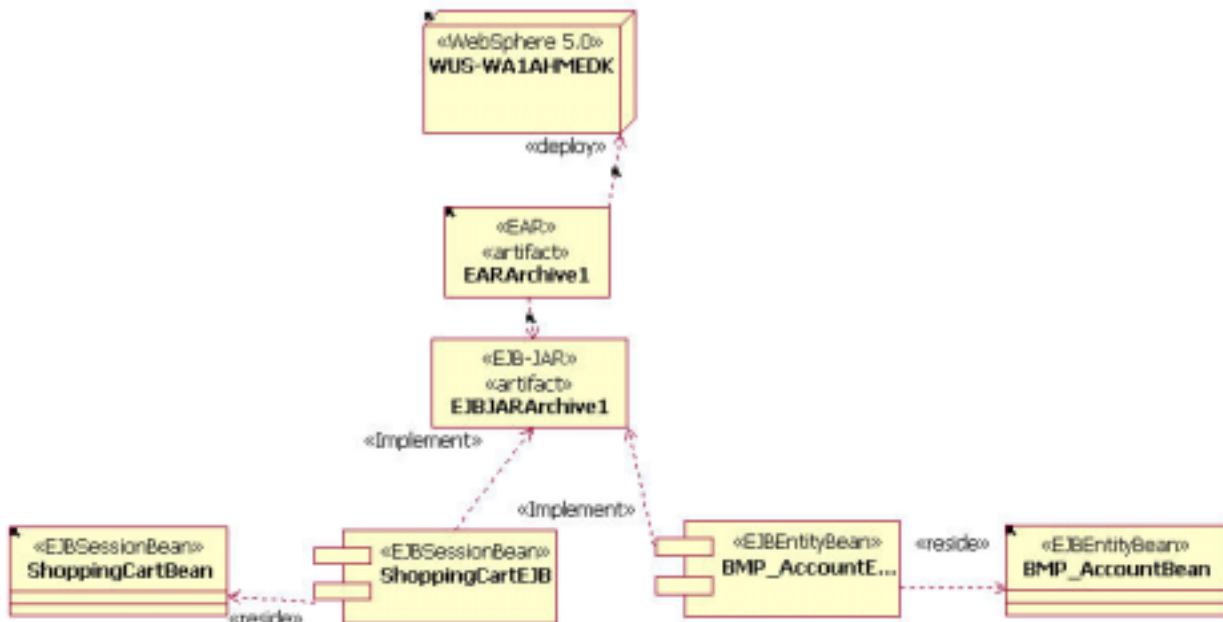


Figure 2. Deployment Modeling — Model n-tier deployment using Rational XDE Developer.






Experience J2EE Design and Development

In this section, you will use the Online Auction example that ships with WebSphere Studio Application Developer Version 5.0 and 5.1.1 to gain experience with the following key Rational XDE Developer features:

- Connector Assistant
- EJB modeling

- Web modeling
- Deployment modeling

Creating the Online Auction Project


1. Click **File > New > Project**.
 2. Select **Examples > Enterprise Applications 1.3** and  Auction . Click **Next**.
 3. Accept the default project name of AuctionExample and click **Finish**.
 4. In the J2EE Navigator, right-click the AuctionRunV5EJB ( AuctionRunV5EJB) project and click **New > Other**.
 5. Select **Modeling** and **Model**. Click **Next**.
 6. Select **Rational XDE > Java** and **EJB Code Model**.
 7. Accept the default names and click **Finish**.
 8. In the J2EE Navigator, double-click the EJB Code Model ( EJB Code Model.mdx).
Rational XDE Developer opens the new EJB code model.
 9. On the main toolbar, depress the **AutoSync** button  to enable auto-synchronization. Once depressed, the button should be colored .
- Note:** For auto-synchronization rules, see **Window > Preferences > Rational XDE > Code-Model Synchronization**.
10. Click **Window > Preferences**.
 11. Expand **Rational XDE > Java > New Element Defaults** and select **Class Defaults**.
 12. Click the **Advanced** tab.
 13. Select the **Create a default constructor method** check box.
 14. Click **OK**.

EJB Modeling

15. In the Model Explorer, right-click the EJB Code Model and click **Reverse Engineer**.
16. A dialog box prompts you for the default location of the source files; click **Yes**.
17. A dialog box prompts you to create a new EJB deployment model; accept the defaults and click **Finish**.




Key Benefit

When synchronizing with an EJB code model, Rational XDE Developer recognizes a deployment descriptor file and prompts you to create the associated EJB deployment model. Using the EJB deployment model, you can easily modify your EJB deployment options and maintain synchronization with the appropriate deployment descriptor file.

18. In the Model Explorer, locate the  (AuctionRunV5EJB) EJB Code Model > com > acme > ejb package.
19. Right-click the ejb package and click **Add Diagram > EJB Diagram**.

Key Benefit



Upon creation of an EJB diagram, Rational XDE Developer populates the diagram with all EJB components detected within the scope of the selected package. EJB diagrams provide a filtered, succinct view of the EJB implementation class. CMP fields are displayed intuitively as attributes. You can filter redundant interface information to improve diagram readability.

20. On the main menu, click **Edit > Select All Shapes**.
21. On the drawing surface, right-click one of the selected shapes and click **Add Related Shapes** .
22. In the **Select In Models** list, click **EJB Code Model**.
23. Click **OK**.
24. On the toolbar, click the **Arrange All Elements** button  to arrange all elements in the diagram.
25. Use the crosshairs  in the bottom-right corner of the diagram to navigate your diagram with a bird's-eye view.
26. Right-click anywhere in the diagram and click **Customize EJB Diagram**.
27. Clear the **EJB References** check box. Click **OK**.
28. On the drawing surface, right-click OnlineitemBean and click **Customize EJB Shape**.
29. Clear the **EJB Interfaces** and **Primary Key** check boxes. Click **OK**.

Key Benefit




Rational XDE Developer EJB diagrams provide complete customization and flexibility, enabling you to communicate only the information you consider critical to your EJB design.

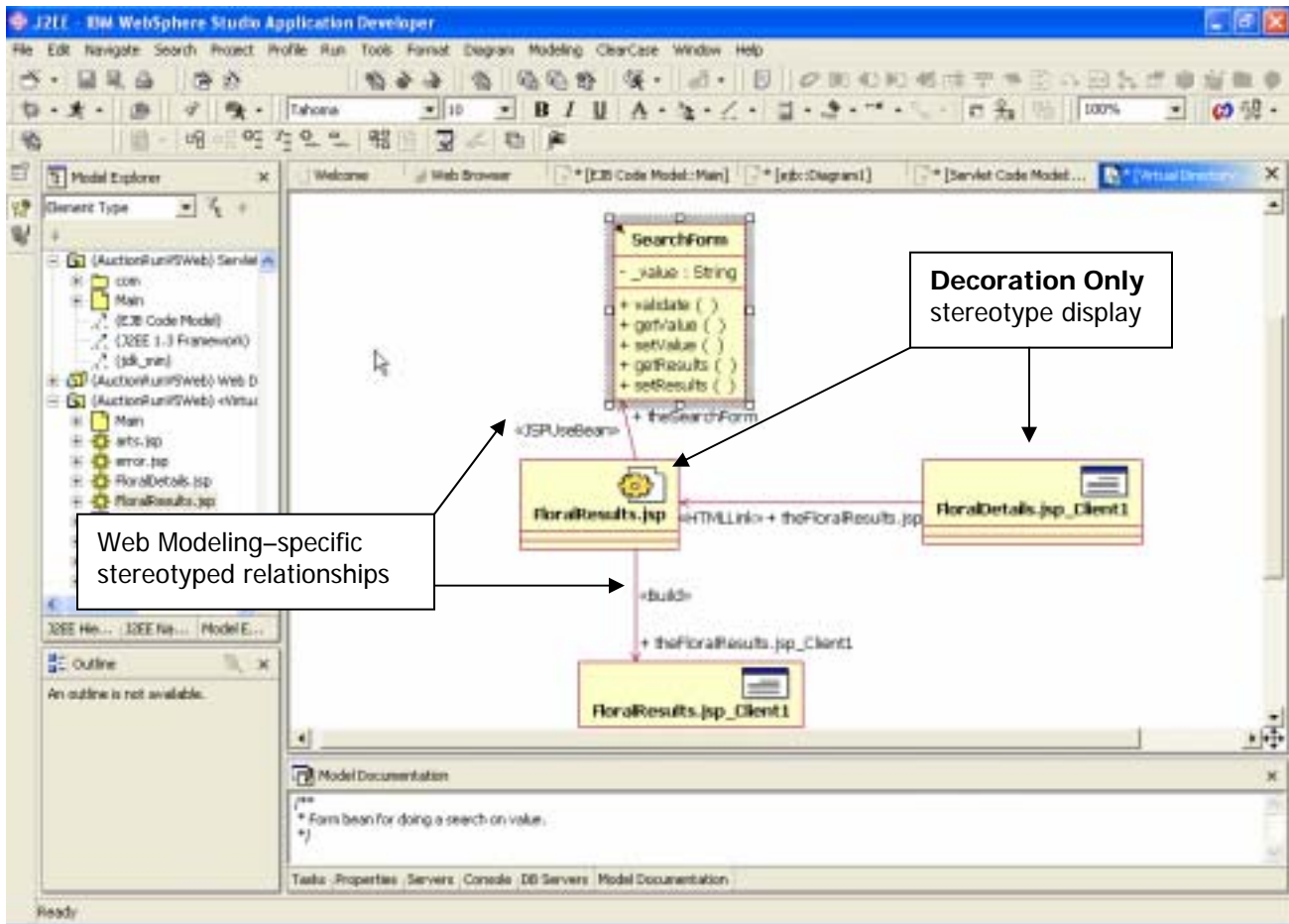
Creating Your Web Models

30. In the J2EE Navigator, right-click the AuctionRunV5Web ( AuctionRunV5Web) project and click **New > Other**.
31. Select **Modeling** and **Model**. Click **Next**.
32. Select **Rational XDE > Java** and **Servlet Code Model**.
33. Accept the default names and click **Finish**.
Rational XDE Developer opens the new servlet code model in the Model Explorer.
34. In the Model Explorer, right-click the Servlet Code Model and click **Reverse Engineer**.
35. A dialog box prompts you for the default location of the source files; click **Yes**.
36. A dialog box prompts you to create a new Web deployment model; accept the defaults and click **Finish**.
37. In the J2EE Navigator, right-click the AuctionRunV5Web ( AuctionRunV5Web) project and click **New > Other**.
38. Select **Modeling** and **Model**. Click **Next**.
39. Select **Rational XDE > Web** and **Virtual Directory Model**.
40. Accept the default names and click **Finish**.
41. In the Model Explorer, right-click the Virtual Directory Model and click **More Java Actions > Add Modeled Files**.

42. Click **Add Recursively**.
43. Click **OK**.

Web Modeling with Rational XDE Developer

44. In the Model Explorer, locate  (AuctionRunV5Web) «VirtualDirectory» Virtual Directory Model > FloralResults.jsp and drag it onto the drawing surface.
45. On the drawing surface, right-click FloralResults.jsp and click **Add Related Shapes** .
46. In the **Select In Models** list, click **All models**.
47. Click **OK**.
48. On the toolbar, click the **Arrange All Elements** button .
49. Click **Edit > Select All**.
50. On the main menu, click **Format > Stereotype and Visibility Style > Shape Stereotype: Decoration Only**.



Key Benefit

Using the **Add Related Shapes** feature of Rational XDE Developer, you can quickly derive all participants for a particular Web site. The resulting diagram visually depicts relationships between Web site artifacts so that you can easily understand how changes might affect the other components. Rational XDE Developer offers various diagram format options, enabling you to configure exactly how to display this information. The relationships are stereotyped with intuitive names like «Build» and «JSPUseBean» so that you can easily understand the flow between the artifacts.

Working with Deployment Models

51. In the J2EE Navigator, right-click the AuctionExample project and click **New > Other**.

52. Select **Modeling** and **Model**. Click **Next**.

53. Select **Java** and **EAR Deployment Model**.

54. Accept the defaults and click **Finish**.

Rational XDE Developer opens the new EAR deployment model in the Model Explorer.

55. In the Model Explorer, right-click (AuctionExample) EAR Deployment Model and click **Reverse Engineer**.

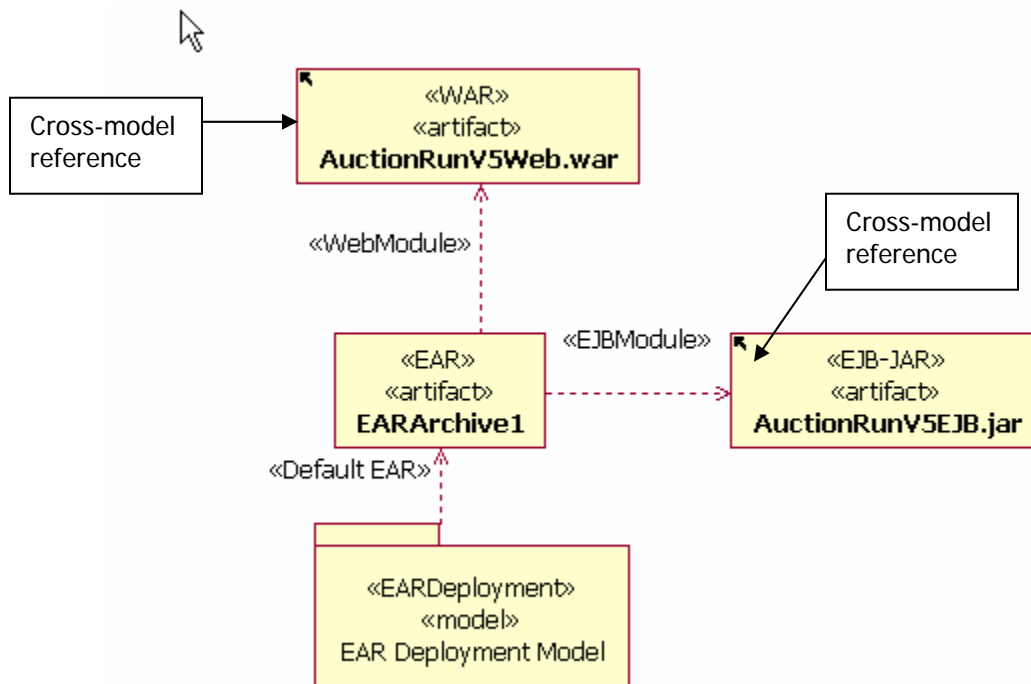
56. Drag (AuctionExample) EAR Deployment Model > EARArchive1 from the Model Explorer onto the drawing surface.

57. On the drawing surface, right-click EARArchive1 and click **Add Related Shapes** .

58. In the **Select In Models** list, click **All models**.



59. Click **OK**.

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



Key Benefit

Rational XDE Developer goes beyond the ability to store your designs in multiple models; it maintains live references across models. Cross-model references enable you to remain in your current model and make updates to elements located in other models, eliminating duplicate work. References across models are indicated visually with a black arrow icon in the upper corner of the model for quick identification. Full search and replace capabilities across models ensure that you can maintain model integrity and consistency without added effort.

61. In the Toolbox, locate the tools in the **EAR Deployment** category and drag a **J2EE Application Server** ( J2EE Application Server) element onto the drawing surface.
62. Click **IBM WebSphere 5.0** in the list and click **OK**.
63. In the Toolbox, click **Connector Assistant** ( Connector Assistant) in the **EAR Deployment** category.
64. Drag from the <<WebSphere 5.0>> Server element to <<EAR>> <<artifact>> EARArchive1.
65. In the Toolbox, drag from **JAR Archive** in the **EAR Deployment** category onto the drawing surface, and name the archive Externallib.
66. In the Toolbox, click **Connector Assistant** again and drag from Externallib to <<EAR>> <<artifact>> EARArchive1.
67. Click **JAR Reference** in the list and click **OK**.

Key Benefit

The Rational XDE Developer Connector Assistant guides you in the creation of relationships between model elements in Java models. You can use the Connector Assistant to determine valid relationships between model elements based on the applied profile of the model. This is especially useful for deployment diagrams when the appropriate relationship is not always obvious.

Benefits of J2EE Design and Development

Rational XDE Developer provides wide support for accelerating and simplifying multitiered J2EE application development. Using UML to unify a project team, Rational XDE Developer supports the specific needs of J2EE development team members, such as project architects, data modelers, application developers, and Web application developers. In particular, it enables J2EE developer and teams to:

- Accelerate EJB component creation using built-in patterns for entity and session beans. EJB-specific diagrams provide a filtered, more intuitive view of your beans so that you can quickly understand relationships and component dependencies. Flexible diagram and formatting options enable you to communicate your design to any project stakeholder.
- Use Web modeling stereotypes to facilitate building Web models and creating relationships between Web artifacts such as servlets, JSP pages, HTML forms, and client pages.
- Easily create intra-model dependencies to link the presentation layer to the business logic and finally to the underlying data design.
- Have an up-to-date view of your entire design, including deployment, that is maintained by Rational XDE Developer. With full deployment modeling capabilities, you can synchronize model changes with actual deployment descriptors.
- Simplify and communicate deployment decisions. Data descriptor information is represented in UML so that you can model EJB, Web, and Enterprise project deployment. Rational XDE Developer offers specialized modeling support in the form of EJB, Web, and Enterprise Archive deployment diagrams;

these diagrams represent, in UML, the state of the deployment descriptors in a visual and easy-to-understand format.

- Create complex EJB-to-EJB or deployment relationships without being a UML expert. The Rational XDE Developer Connector Assistant removes the guesswork from creating relationships between model elements, saving you precious time that would otherwise be wasted on trial and error.

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 program behavior and communicate it to 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

Opening the Project and Enabling Visual Trace

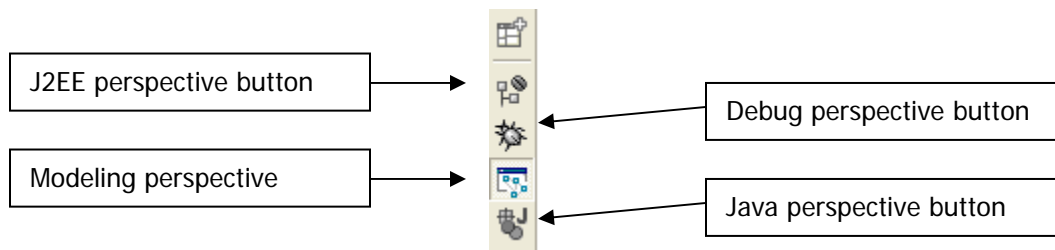
1. In the Navigator, right-click the CardApp project and click **Synchronize**.
2. On the main menu, click **PurifyPlus > Visual Trace > Engage Visual Trace**.
3. In the Visual Trace Settings dialog box, expand the **CardApp > (default)** folder and add the following classes from the list: BlackjackHand, Card, Deck, and Hand.
4. 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 JDK Framework classes used by your application to understand how your application interacts with those components.

Visually Tracing the CardApp Project

5. On the perspective toolbar, click the Java perspective button to switch to that perspective.

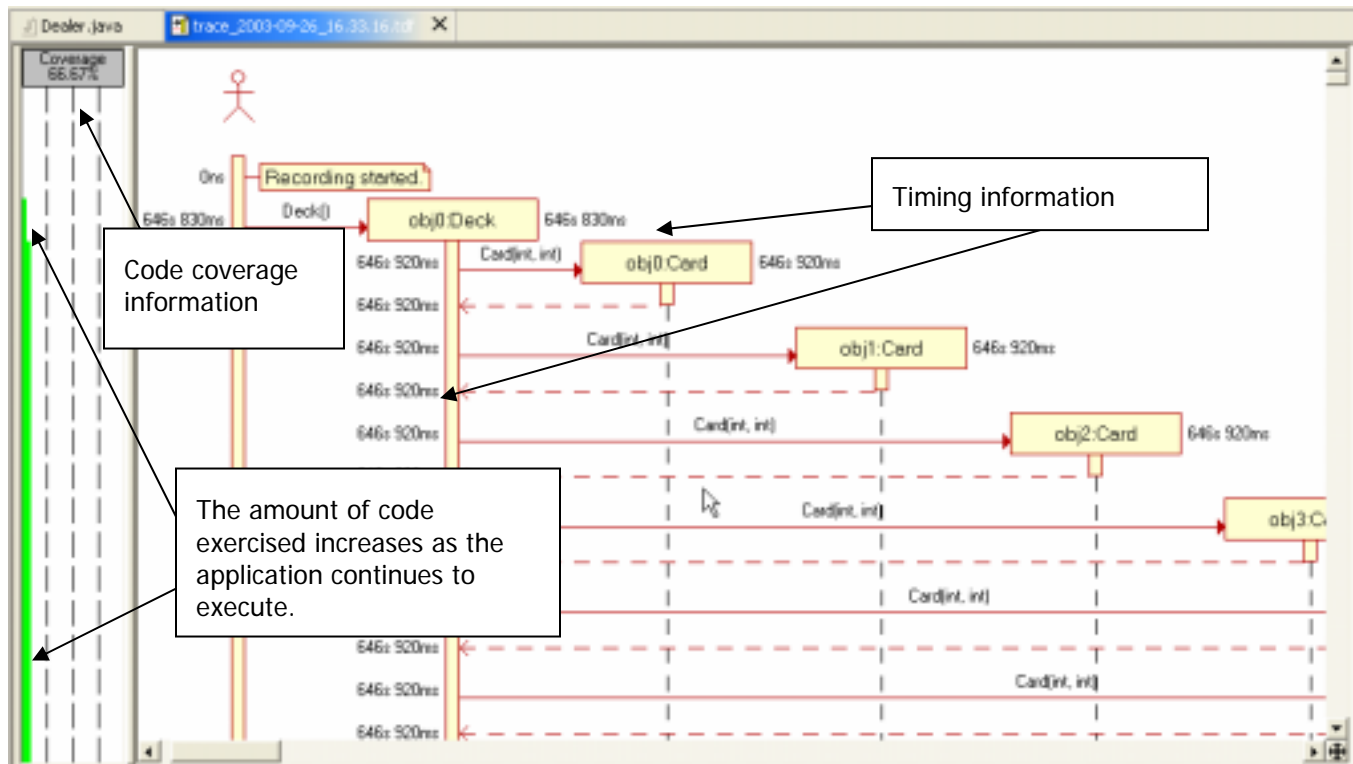


You can also open the Java perspective from the main menu, by clicking **Window > Open Perspective > Java**.

6. In the Package Explorer, select the CardApp project.

7. On the main menu, click **Run > Run As > Java Bean/Applet**.
8. Select BlackjackConsole from the list and click **OK**.
9. When the Blackjack application begins, click the **Run the Program** button.
10. Follow the application instructions to wager a bet and play a hand of Blackjack.
11. To end the game, type 0 and press Enter.

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 Visual Trace Diagram

12. Click the Modeling perspective button on the perspective toolbar to return to the Modeling perspective.
13. Scroll to the top of the trace diagram and slowly scroll down to view the 52 cards being created.
14. Right-click in the trace diagram and click **Show Trace Tree**.
15. In the Trace Tree window, collapse the Card object list.
This will simplify the diagram by showing only a single Card object lifeline.
16. Right-click in the trace diagram and click **Hide Trace Tree**.
17. In the trace diagram, right-click the Card(int,int) message and click **Filter Message**.

Key Benefit

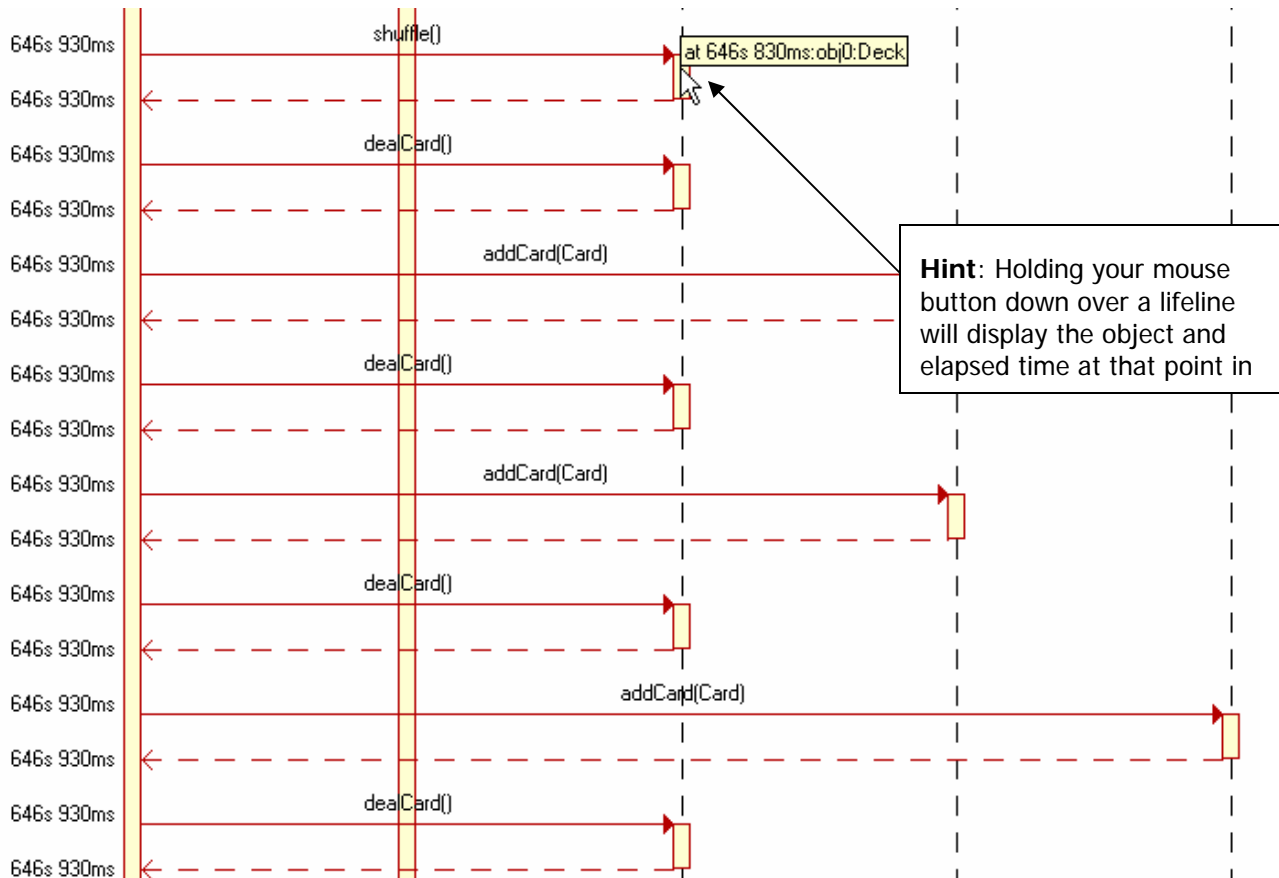
Trace diagrams come with a number of formatting options to help you identify key application behavior quickly. You can easily filter specific messages, individual objects, and object types that are not relevant to the specific program behavior you are trying to capture and diagnose. To facilitate iterative 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

18. Toward the top of the trace diagram, locate the `BlackjackHand()` message and the `obj0:BlackjackHand` and `obj1:BlackjackHand` objects. (Here the application creates two hands, one for the player and one for the dealer.)



19. Scroll down slightly in the diagram and locate the `Shuffle()` message. Note that each time the user chooses a hit, the `Deck` class deals a card, which is then passed to the `obj0:BlackjackHand` object from the main application. The same pattern is repeated for the `obj1:BlackjackHand` object as the dealer tries to beat the current hand.



20. Scroll down in the trace diagram, locate the `getBlackjackValue()` message. This seems like a critical piece of application functionality, because the actual value of `BlackjackHand` is calculated to determine whether the dealer or the player wins. Right-click the `getBlackjackValue()` message and click **Browse Code**.

Key Benefit

Once you have identified critical program functionality in a trace diagram, you can easily toggle between a high-level view of your application and source-level details.

21. Right-click in the trace diagram and click **Generate Sequence Diagram**.
22. In the **New package to contain sequence diagram** text box, rename Diagram to CardApp Trace.
23. Click **Finish**. You may need to filter additional messages or lifelines if the trace diagram is too large.

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. While this exercise demonstrated Visual Trace using a simple Java application, you can also capture trace information for your J2EE components running in the WebSphere test environment.

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.

Development with Custom Patterns and Code 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, ensure efficiency and quality throughout the lifecycle, and encourage code standardization for more consistent, reliable project growth.

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, better yet, 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 reusing proven code.

In short, designing your own patterns in Rational XDE Developer enables you to jump-start your development, fully use your creative capabilities, avoid repetitive, tedious tasks, and minimize mistakes.

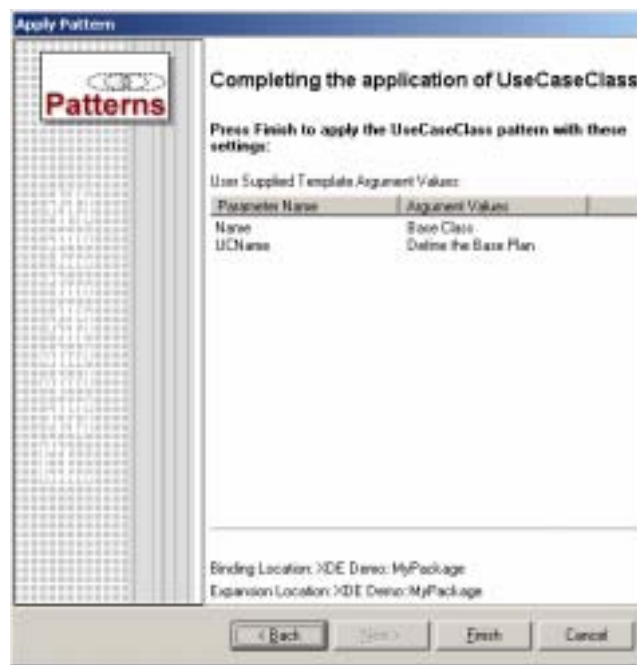


Figure 3. Patterns — Create and apply patterns to maximize your efficiency and effectiveness and improve your software quality.

Development with Custom Patterns and Code 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, ensure efficiency and quality throughout the lifecycle, and encourage code standardization for more consistent, reliable project growth.

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, better yet, 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 reusing proven code.

In short, designing your own patterns in Rational XDE Developer enables you to jump-start your development, fully use your creative capabilities, avoid repetitive, tedious tasks, and minimize mistakes.

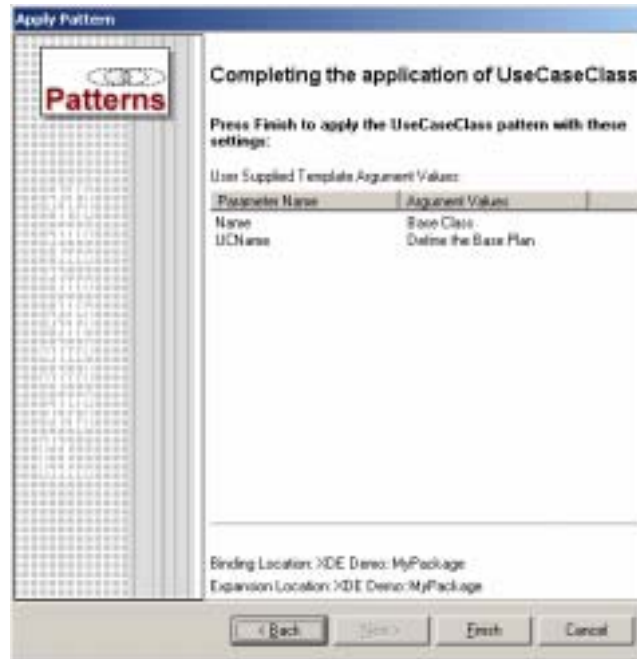


Figure 4. Patterns — Create and apply patterns to maximize your efficiency and effectiveness and improve your software quality.

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.

To ensure that your system is in the correct state for this exercise, first perform the steps below to disable auto-synchronization:

1. On the main menu, click **Window > Preferences**.
2. Expand **Rational XDE > Code-Model Synchronization**.
3. Clear the **AutoSync** check box.

- Click **OK**.

Creating a Simple Project for Storing Pattern Models

- Click **File > New Project**.
- Select **Simple** and **Project**. Click **Next**.
- Name the project MyPatternProject and click **Finish**.

Creating a Model for Storing Pattern Models

- In the Navigator, right-click the MyPatternProject project and click **New > Other**.
- Select **Modeling** and **Model**. Click **Next**.
- Select **C++** and **C++ Content Model**.
- Name the model file MyPatternModel and click **Finish**.

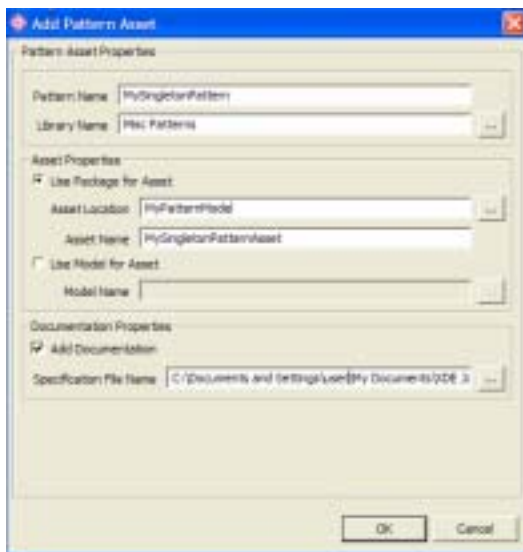
The Model Explorer displays the MyPatternModel model with its Main diagram.

Creating a New Pattern

- Click **Window > Open Perspective > Modeling**.
- 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.



- In the Add Pattern Asset dialog box, set the **Pattern Name** to MySingletonPattern.
- Under **Asset Name**, replace RASPackage1 with MySingletonPatternAsset.



- Leave the defaults in the remaining fields and click **OK**.

Key Benefit

Rational XDE Developer assets consist of models written and packaged to be reusable. The structure of assets is based on the Reusable Asset Specification (RAS). Pattern assets in particular assist in pattern exchange, facilitate user application of patterns, and provide pattern documentation.




17. 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 enables you to 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**.

18. In the Model Explorer, right-click MySingletonPattern ( MySingletonPattern) and click **Add UML > Template Parameter**.
19. With TemplateParameter1 selected, enter InputClass to rename the template parameter. Click elsewhere to deselect the template parameter name.
20. 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.
21. Rename the default name to PatternClass.
This class will define the results of applying MySingletonPattern to classes.
22. Right-click PatternClass and click **Add UML > Attribute**. Leave the default name as Attribute1.
23. With Attribute1 ( Attribute1) selected in the Model Explorer, go to the Properties view and set the following UML properties for this attribute:

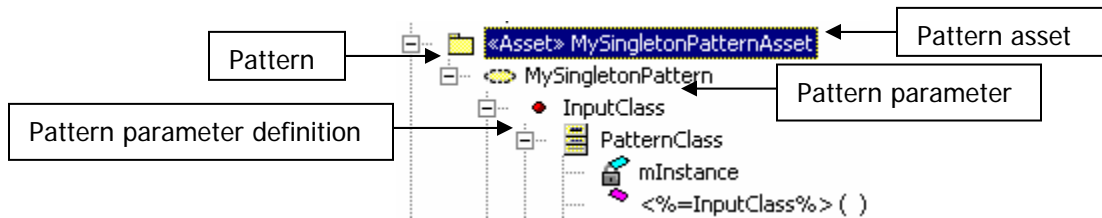
Name	mInstance	Attribute name
DefaultValueExpression	null	Maps directly to C++ ; be sure to use lowercase.
OwnerScope	CLASSIFIER	Attribute qualifier (maps to C++ static modifier; applied to field)
TypeExpression	<%=InputClass%>	Scriptlet (see explanation below)
Visibility	PRIVATE	

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.

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

Using the same substitution mechanism, when the pattern is applied Rational XDE Developer will create a new constructor for the class to which the pattern 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.

25. Right-click in the Navigator and click **New > Project**.

26. Select **Modeling** and **C++ Modeling Project**. Click **Next**.

27. Name the project `MyPatternTestProject` and click **Finish**.

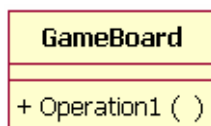
(`MyPatternTestProject`) C++ Code Model is added in the Model Explorer.

28. Double-click in the  Main diagram in the `MyPatternTestProject` model to make it the active diagram.

29. Right-click on the drawing surface and click **Add C++ > Class**.

30. Name the class `GameBoard` and leave all other defaults. Click **OK**.

31. Right-click the `GameBoard` class and click **Add C++ > Operation**. Keep the defaults and click **OK**.



Applying Your Pattern

You can now apply your `MySingletonPattern` pattern to 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).

32. In the Model Explorer, right-click the `MySingletonPattern` pattern ( `MySingletonPattern`) and click **Add to Pattern Favorites** ( `Add to Pattern Favorites`).

Key Benefit

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

33. On the drawing surface, right-click the `GameBoard` class and click **Apply Favorite Pattern**.

34. 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**.

35. 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.

36. Click **Next**.


Key Benefit

The Apply Pattern Wizard provides an easy way to reuse patterns. By simply entering pattern parameters via a succession of wizard screens, you can accelerate your development and ensure higher-quality code. As you become more advanced, using the single dialog interface may prove to be a more convenient way of applying complex patterns.

37. 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. Click **Next**.

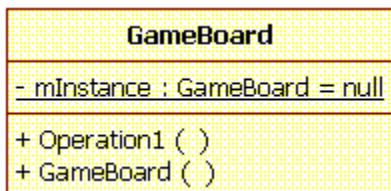
Key Benefit

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.

38. On the **Expansion Location** screen, click  (MyPatternTestProject) C++ Code Model and click the **Select** button to specify your MyPatternTestProject project as the project in which to expand the pattern.

39. 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; 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**).

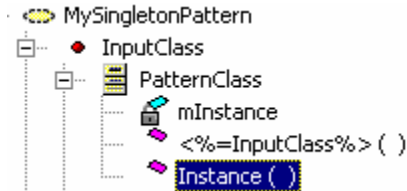
Binding Code Templates to Patterns

Now that you have validated that your MySingletonPattern pattern works as expected, you will include some reusable code in it by attaching a custom code template to the Instance operation of MySingletonPattern. You can bind code templates to operations to specify code to be generated in the corresponding method bodies.

Key Benefit

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

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



41. 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.

42. With Parameter1 selected in the Model Explorer, go to the Properties view 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).

43. 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.
44. In the Bind Code Template dialog box, click **New**. The Create Code Template Wizard appears.
45. In the **Name** box on the first of the wizard's screens, enter MyFirstCodeTemplate.
46. Optionally enter some descriptive text for the code template in the **Description** box. Click **Next**.
47. On the **Step 2 of 2** screen, click **Add**.

Fill in the Template Parameter dialog box as follows:

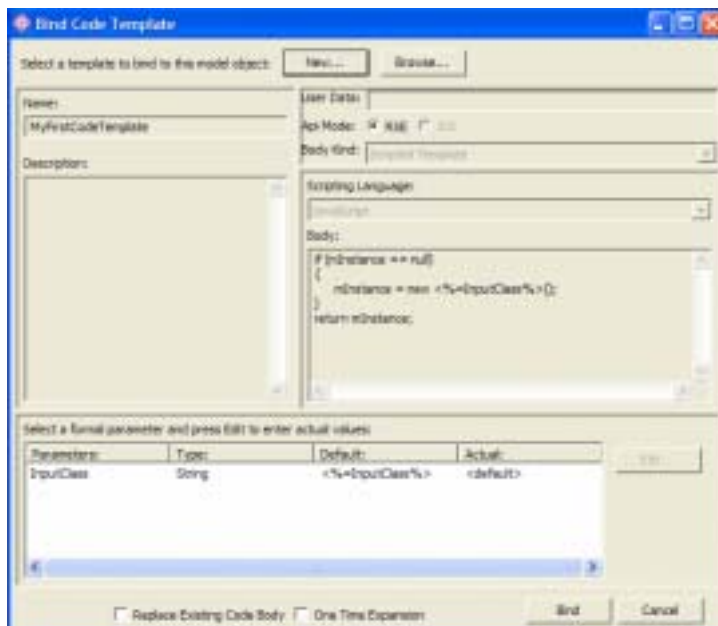
Name	InputClass	The InputClass template parameter will be used by the code template to get information.
Type	String	The value extracted from the InputClass template parameter will be of type String.
Default	<%=InputClass%>	The default value will be the result of the <%=InputClass%> scriptlet evaluation,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

48. Click **OK**.

49. In the **Body** text box, enter the following code:

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

50. Click **Finish**.



51. 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. Using proven patterns and automating mundane coding through code templates saves you time. 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.

52. In the Model Explorer, scroll down to MySingletonPattern_Binding (  MySingletonPattern_Binding).
53. 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 dialogs to view your previous input or just click **Finish** to apply the pattern.

54. Click **Next**; then click **Next** again.
55. 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 ()

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

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.

57. 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 Source**. The code appears as follows:

```
#include "GameBoard.h"

/**ModelId={942169B3-62E1-473B-ABC9-28DC9F2E5906}
void GameBoard::Operational()
{
}

/**ModelId={C28F4DDB-DE51-4C17-88FE-72F3E3C9428D}
GameBoard::GameBoard()
{
}

/**ModelId={AA69C6D7-FDB0-4A69-8D40-65ADAA261E2D}
GameBoard GameBoard::Instance()
{
    /*Begin Template Expansion{8B3D7B65-E96F-4288-B830-4E4B5C22CFA4}*/
    if (mInstance == null)
    {
        mInstance = new <%=InputClass%>();
    }
    return mInstance;
    /*End Template Expansion{8B3D7B65-E96F-4288-B830-4E4B5C22CFA4}*/
}

/**ModelId={398D9246-6CDB-4556-8469-64555B4443DE}
GameBoard GameBoard::mInstance = null;
```

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.

58. 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. Its pattern support is flexible in various ways:

- Via a binding between the pattern and its expansions, Rational XDE Developer enables you to 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 enables you to 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 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 repetitive and tedious 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.

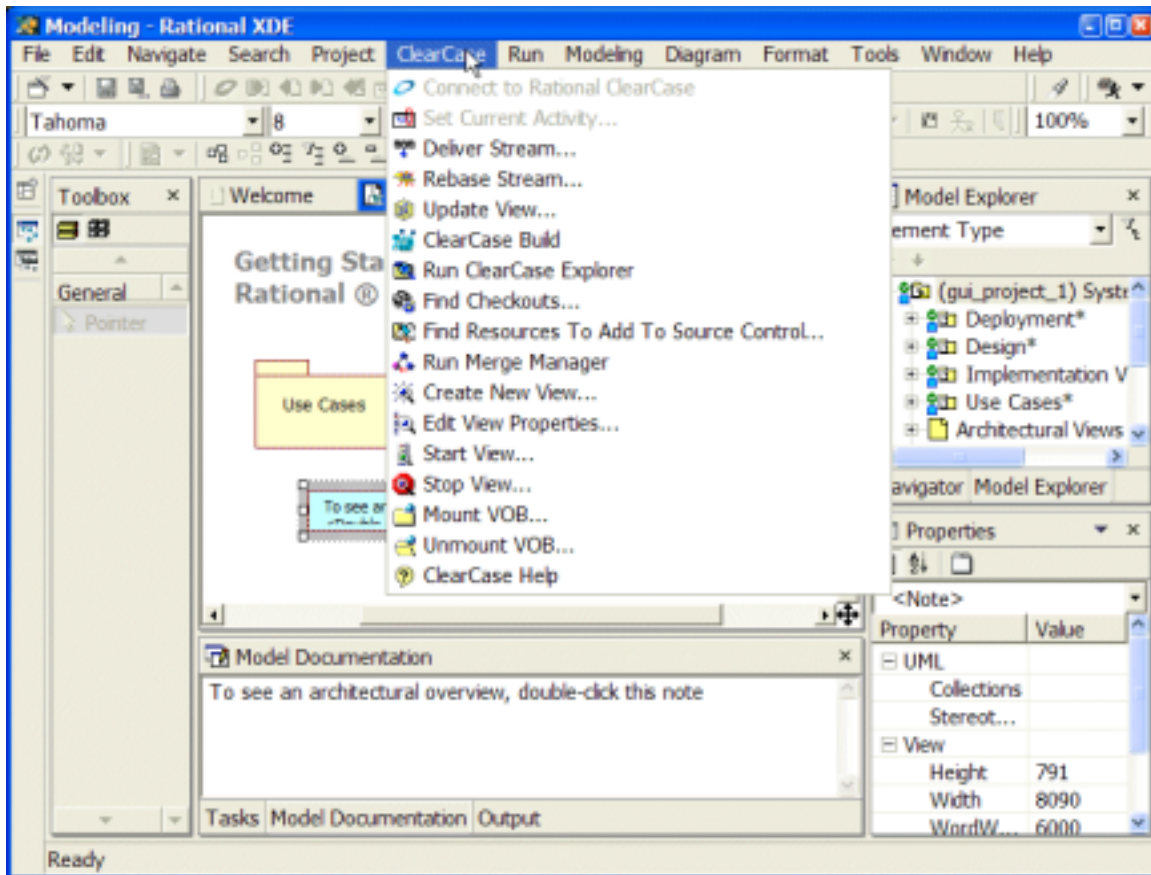


Figure 5. Rational XDE Developer and Rational ClearCase Integration — Configuration management access directly through the development environment.

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.

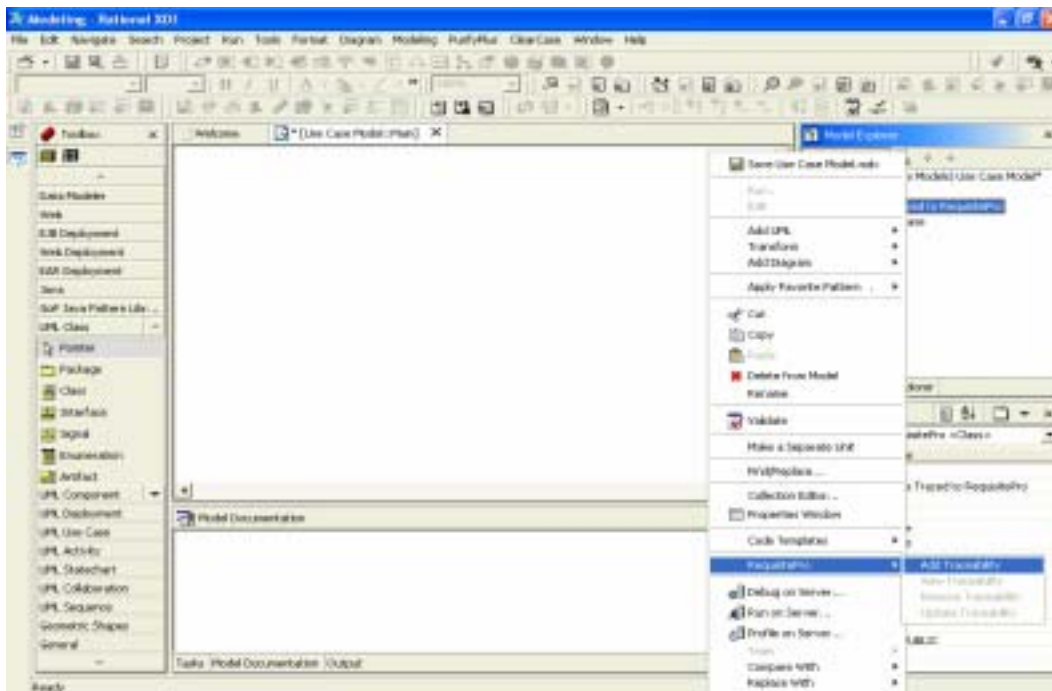


Figure 6. Rational XDE Developer and Rational RequisitePro Integration — Trace design elements to requirements by integrating with RequisitePro.

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 the Rational area of the DeveloperWorks website (<http://www.ibm.com/developerworks/rational/>). The DeveloperWorks Web site 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 DeveloperWorks are:

- A Rational XDE Knowledge Center, where you will find all content related to Rational XDE Developer (including specific threads addressing the needs of developers using the J2EE platform)
- A Rational XDE Developer tutorial
- Introductory technical articles written by J2EE 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

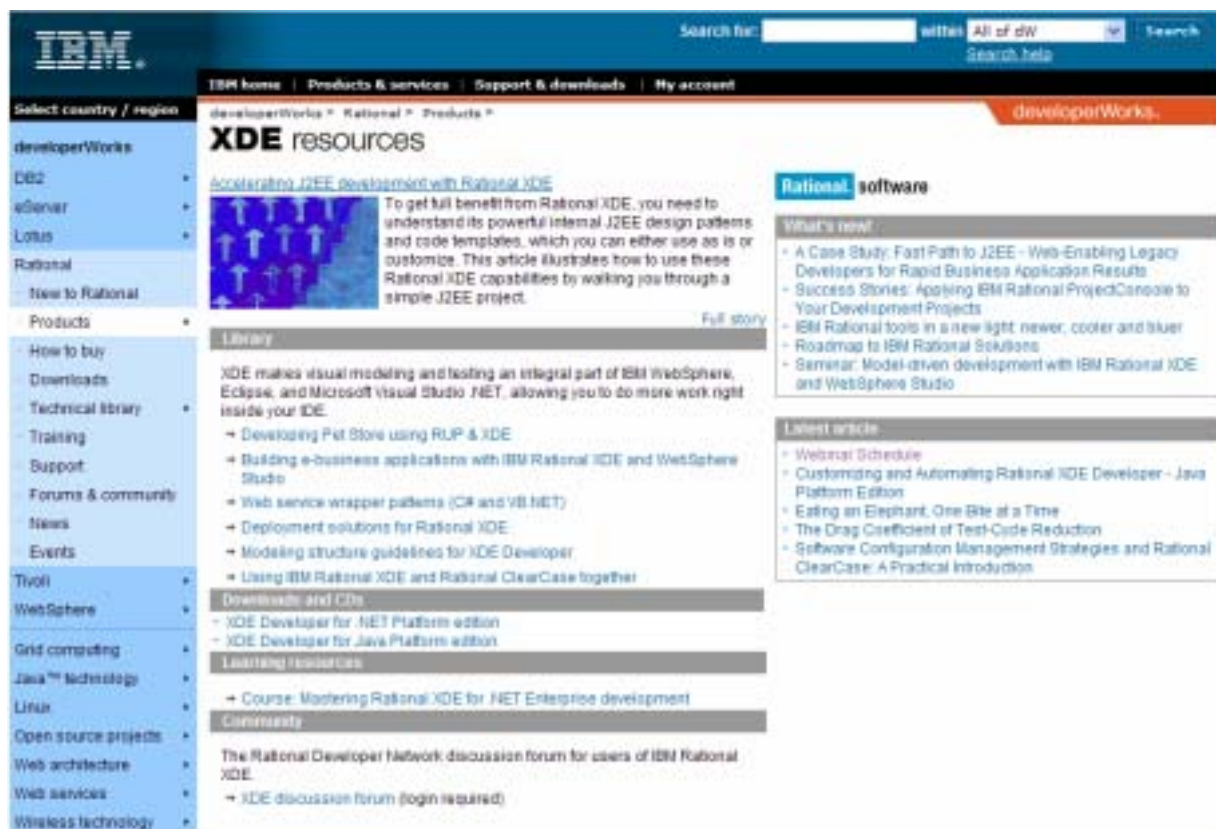


Figure 7. DeveloperWorks — Additional resources to further your development efforts.

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 (which can be accessed through DeveloperWorks at <http://www.ibm.com/developerworks/rational/downloads/>) 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 Java-platform 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 packaged in the online help, available from Help.
- The Rational XDE Developer product page at <http://www.ibm.com/software/awdtools/developer/rosexde/>.
- If you are an IBM Rational customer with an active maintenance agreement, you can explore the Rational XDE Center on DeveloperWorks, where you will find reusable assets (select **Downloads > Assets and Patterns Exchange**), Java training, and J2EE process guidance (select **Downloads > RUP Plug-In Exchange**).

Appendix: Rational XDE Developer Interface Overview

This appendix provides a quick visual overview of the Rational XDE Developer interface and the various user interface elements referred to in this guide. For a deeper look, you can consult the *Workbench User Guide* by clicking **Perspective > Open > Other** in Rational XDE Developer, selecting the **Help** perspective, clicking **Workbench User Guide** in the list of choices, and selecting **Getting Started > Basic Tutorial**.

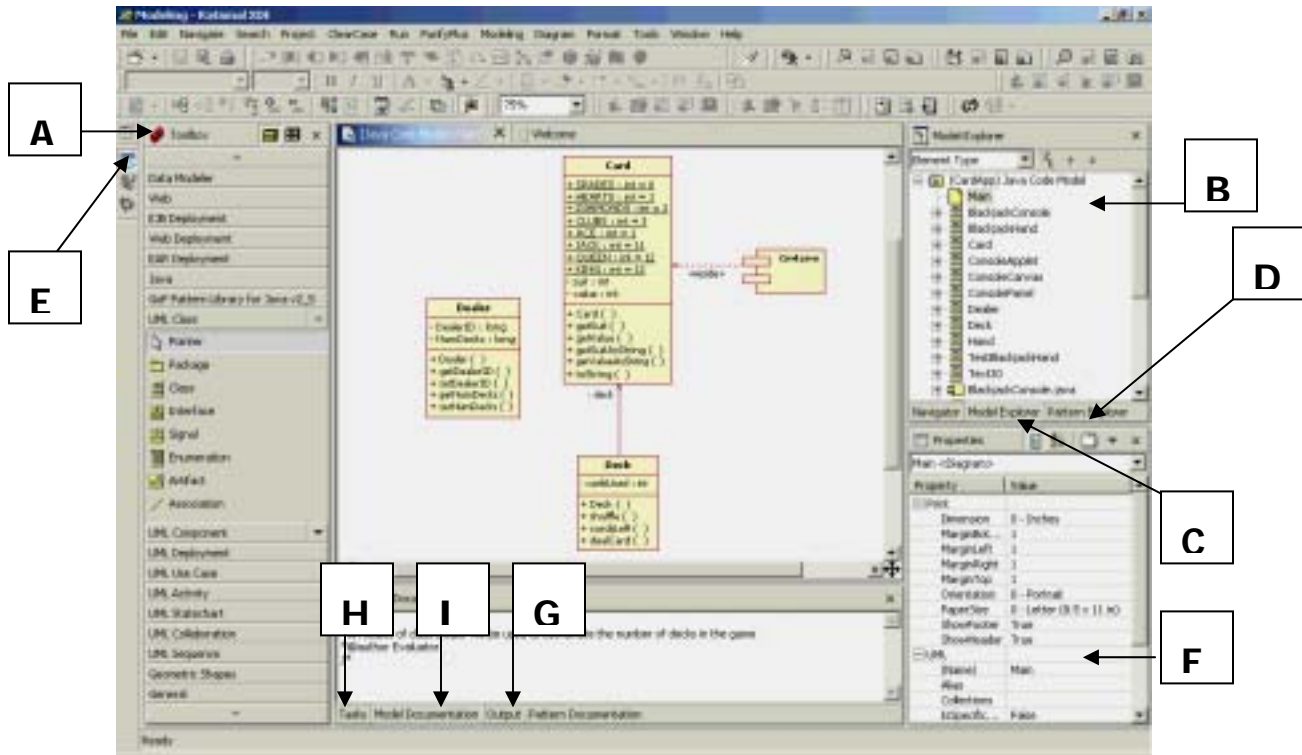


Figure 8. Rational XDE Developer Main Interface — A single interface for modeling and developing Java applications.



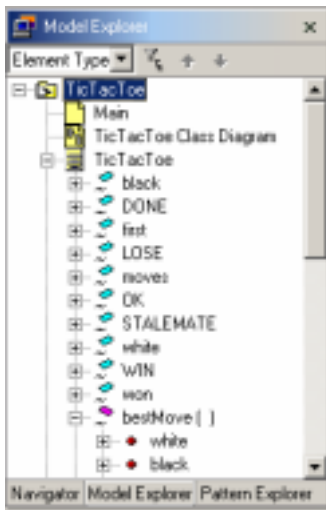
A. Toolbox

The Toolbox is a 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. Navigator

The Navigator provides a code-centric view of a project. Multiple projects can be viewed at any time. A project stores a collection of files that can include models, source code, storage units, text files, and other project-related artifacts.

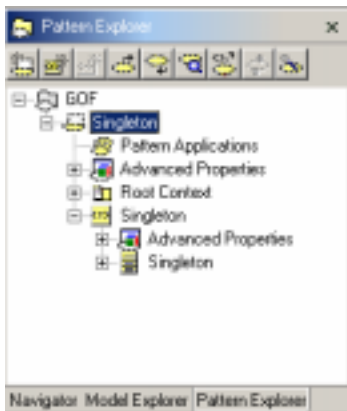


C. Model Explorer

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.

This window is stacked with the Navigator and Pattern Explorer windows.

Similar to the Navigator, the Model Explorer enables the user to view and modify multiple models at any given time.



D. Pattern Explorer

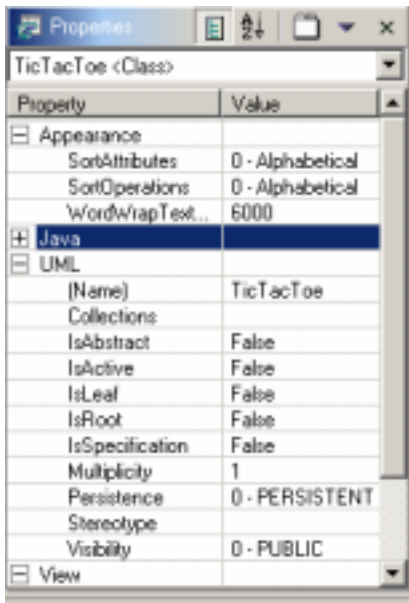
The Pattern Explorer displays all elements that are used by a specific pattern.

Similar to the Navigator, the Pattern Explorer enables the user to view and modify multiple patterns at any given time.



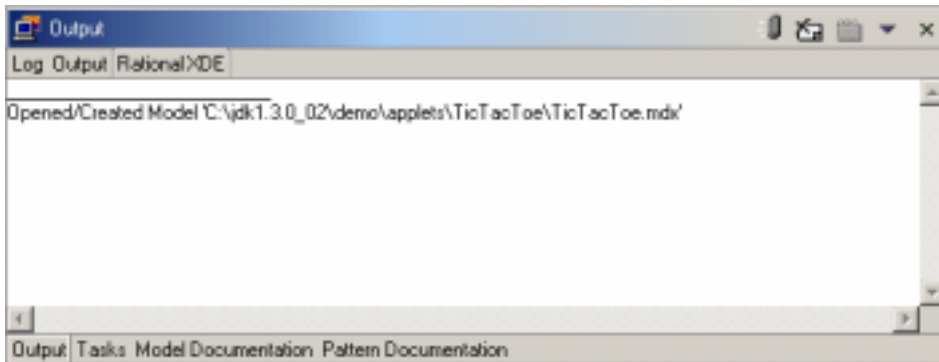
E. Perspectives

Perspectives provide a way to customize the initial layout and selection of windows typically used for a given task. Users can easily switch from one perspective to another to carry out different tasks. Common perspectives include Modeling, Debug, Help, and Java. Users can also create their own custom perspectives.



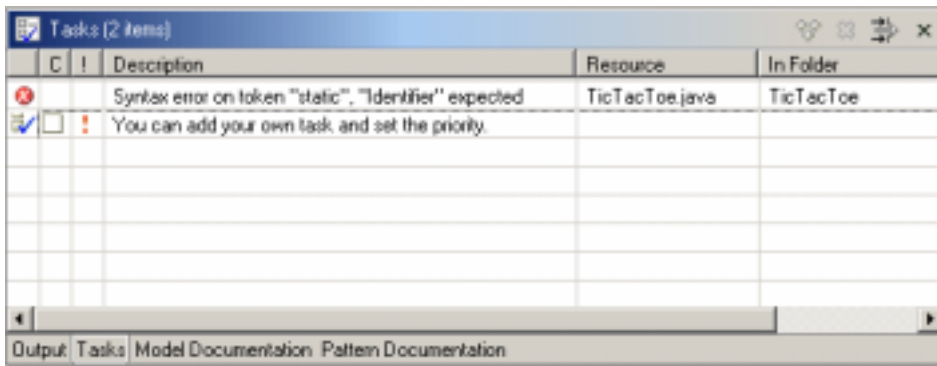
F. Properties View

The Properties view is context-sensitive and will display all the given properties for a selected element. These properties can be modified from within the Properties view. When you create patterns, a Properties view specific to pattern creation is opened.



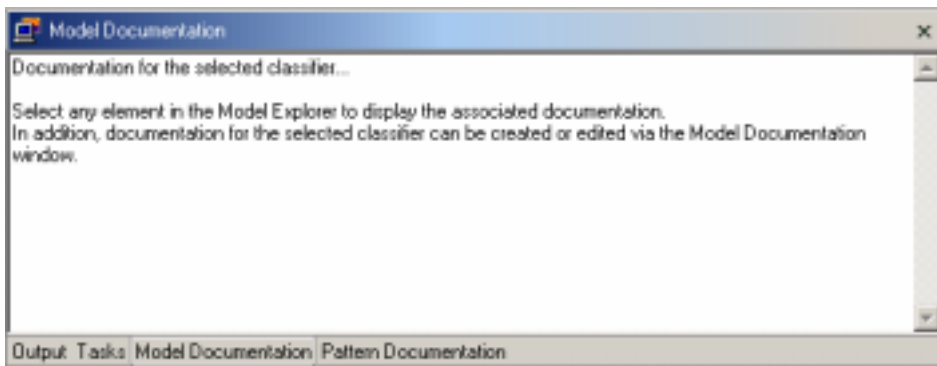
G. Output Window

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, or model element.



H. Tasks Window

The Tasks 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 Tasks window will display compilation and build errors and warnings; when you are viewing model diagrams, model validation errors and warnings will be displayed.



I. Model Documentation Window

The Model Documentation window displays documentation at the model element level. This window is stacked with the Output and Tasks windows.