Rational Rose 2000 Quick Start with Rose Visual Basic

Contents

List of Figures

Preface

Use the tutorial in *Quick Start with Rose Visual Basic* to become quickly acquainted with the tool. You will be introduced to some basic features in Rational Rose Visual Basic, and learn how to:

- Understand an application by browsing existing design diagrams and Specifications.
- Modify an application by modeling the changes.
- Implement changes to an application by generating and evolving the code and updating the design.
- Understand how the design model is mapped to Visual Basic code.

Prerequisites

To use this tutorial effectively, you should be comfortable with basic Microsoft Windows techniques, including using a mouse and choosing from menus. You should also be familiar with the Microsoft Visual Basic programming environment.

To run the example project, you need to have Microsoft Visual Basic installed on your system.

To browse in the example model you need Rational Rose with the Visual Basic Language Support add-in installed on your system.

In the tutorial, the Component Object Model (COM) terminology is used when referring to properties and methods. However, by default Rational Rose uses the Unified Modeling Language (UML) terminology to refer to those terms—that is, properties are called attributes and methods are called operations in Rational Rose. If you want Rational Rose to use the COM terminology, do as follows:

1. Exit Rational Rose.
2. Open rose.ini, which is located in the Rational Rose installation folder.
3. Search for UseCOMTerminology and set it to "Yes." Save the file.
4. When you restart Rational Rose, the COM terminology is used in all specifications, dialog boxes, and menu items.

How this Tutorial Is Organized

This tutorial contains the following exercises:

- **Chapter 1**— Introduction to the Order System Example

  In the first part of the tutorial you will be introduced to the order system application by running the current implementation.
- **Chapter 2** — Browsing the Use-Case Model

  In this exercise, you will use Rational Rose to get acquainted with the use-case model of the order system. You will also get an overview of how the objects in the design model interact in order to perform different scenarios of the use cases.
- **Chapter 3** — Browsing the Order System Design Model

  In this exercise, you will use Rational Rose to look at the architecture of the order system's design model. You will also see how the model is prepared for code generation.
- **Chapter 4** — Round-Trip Engineering with Rational Rose Visual Basic

  The last part of the tutorial introduces you to round-trip engineering as you are going to create a new class in the model and generate code from it. Finally, you are going to reverse engineer some code changes into the model.

Reading Instructions

It is best to complete the parts of the tutorial sequentially. If you choose to do only a part of the tutorial, you should start with the first section in one of the chapters. For example, you can skip the entire

Introduction to the Order System Example chapter and start with chapter *Browsing the Use-Case Model*, but you should not jump directly into the middle of a chapter.

File Names

The sample used in this tutorial is a design model of an order system and the corresponding Visual Basic code. The files that belong to the sample are located in the **`Samples\Ordersystem`** folder in your Rational Rose installation folder.

Online Help

Rational Rose 2000 includes comprehensive online help with hypertext links and a two-level search index.

Online Manual

Rational Rose 2000 includes all the user manuals online. Please refer to the Readme.txt file (found in the Rational Rose installation folder) for more information.

Chapter 1

Introduction to the Order System Example

Imagine yourself as the newest member of the MIS department in a small fish and seafood distributing company. Your first assignment is to make changes to the company's sales order support application. After meeting with your boss, you understand that:

- An order has a purchaser
- An order consists of one or more order rows, each of a given article in a given quantity
- Orders are registered and handed to shipping
- Customers, orders, and articles need to be persistently stored by the order system

Before making any decisions on how to change the order system, you wisely decide to run the current implementation and to use Rational Rose for studying the current architecture.

The steps to run the example are:

1. Starting the order system
2. Entering a new order

Starting the Order System

1. Start your Microsoft Visual Basic development environment by double-clicking on the **`ordersys.vbp`** project file. You can find this file in the **`Samples\Ordersystem`** folder in your Rational Rose installation folder.
2. Run the program by clicking **Run > Start With Full Compile**. The following dialog box appears:

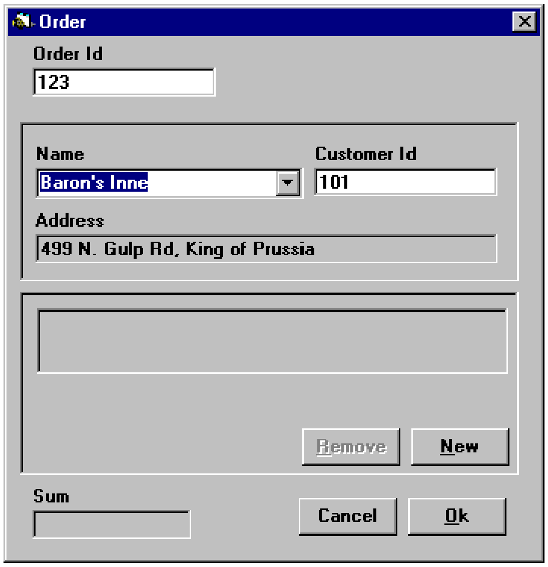


Figure 1 Order Dialog Box

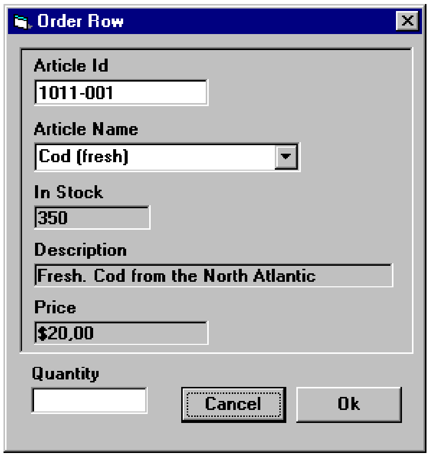
If you have problems starting the order system, make sure that:

- The database files **`ordersys.mdb`** and **`ordersys.ldb`** are located in the same folder as the project file, and that the files are writable (that is, not read-only).
- The DAO Object Library is loaded. Click **Project > References** and select the appropriate library.
- None of the references in the **References** dialog box are marked as "**MISSING**."

Entering a New Order

After having successfully started the order system application you are now ready to enter a new order. An order ID has already been allocated in the **Order** dialog box.

1. Start by assigning a customer to the new order. By default the first customer found has been chosen, which is “101,” “Baron's Inne,” at “499 N. Gulp Rd, King of Prussia.” Click the arrow in the **Name** box to view all other customers and select one of them.
2. Click **New** to open the **Order Row** dialog box.



Article Id: 1011-001
Article Name: Cod (fresh)
In Stock: 350
Description: Fresh. Cod from the North Atlantic
Price: $20,00
Quantity:

Figure 2 Order Row Dialog Box

3. Select an article from the **Article Name** box and see how the information in the dialog box changes.
4. To populate the **Article Name** list with seafood articles with a higher article ID, type “2*” in the **Article ID** box and press ENTER.
5. Select any article, type a number in the **Quantity** box, and click **OK.**
6. The **Order** dialog box now presents the new order row and recalculates the total order sum after each new order row has been created.
7. Click **OK** to accept and register the new order.
8. Click **OK** to exit the order system.

Chapter 2

Browsing the Use-Case Model

In this exercise, you will use Rational Rose to get acquainted with the use-case model of the order system. You will also get an overview of how the objects in the design model interact to perform different scenarios of the use cases. The tutorial takes you through the following exercises:

1. Opening the order system model
2. Browsing the use cases
3. Connecting an external document to a use case
4. Viewing the scenarios
5. Mapping the scenarios to Visual Basic code

Use cases are used throughout the entire development cycle as follows:

- In the requirements analysis phase, use-case modeling plays an important role for capturing and documenting user requirements for the system. Use-case modeling then focuses on user interaction with the system and what the system is supposed to do.
- During design, use cases are used to distribute the requirements of behavior among objects and to show how the objects interact to perform different scenarios of the use cases. The interaction is illustrated in sequence diagrams.
- Finally, the use cases are used during testing as test cases. They are also important input when writing the user documentation of the system.

Opening the Order System Model

1. Start the Rational Rose application.
2. If the Framework Add-In is installed and enabled, the **Create New Model** dialog box is shown, where you click the **Existing** tab. If the Framework Add-In is not enabled, click **File > Open**.
3. Open the **`Samples\Ordersystem`** folder, which is located in your Rational Rose installation folder.
4. Double-click on the **`ordersys.mdl`** file to open the sample model.
5. On the left side of the application window, a browser window lists the contents of the order system model. (If you cannot see the browser in your Rational Rose application window, click **View > Browser**.)
6. As you can see in the browser, a system is described from different views: use-case view, logical view, component view and deployment view. The logical view is the actual design model of the system, which you will examine later in the tutorial.
7. Expand the use-case view by clicking the “+” sign next to it. The different symbols mean:

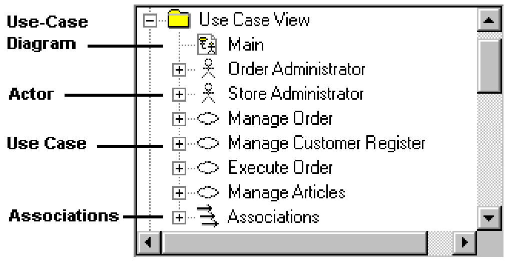


Figure 3 Expanded Use-Case View in the Browser

8. A documentation field is located below the browser. This field contains a textual description of the currently selected model element or diagram. (If the documentation field is not displayed, click **View > Documentation**.) Select an actor in the browser and take a look at its description.

Browsing the Use Cases

1. Open the use-case diagram called Main by double-clicking on it in the browser.

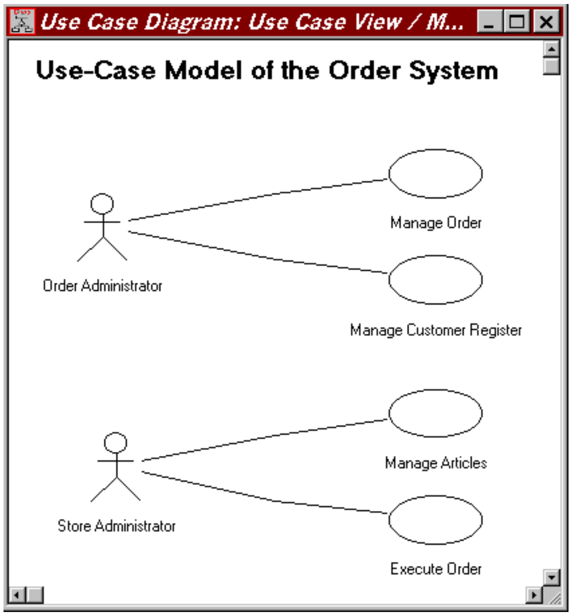


Figure 4 Main Use-Case Diagram

2. As you can see, the use-case model of the order system consists of two actors (Order Administrator and Store Administrator) and four use cases (Manage Order, Manage Customer Register, Manage Articles, and Execute Order).
3. Double-click on the Manage Order use case to open its Use-Case Specification, and take a look at the textual description in the **Documentation** box.
4. Close the specification.

Connecting an External Document to a Use Case

To make it easy to understand the use cases, their flow of events are best described using plain text. As you know by now, the flow of events for the order system use cases are described in the **Documentation** box of their specifications. However, if the description of a use case becomes large or complicated, it can be documented in an external file instead and connected to the use case. To connect a file to the Manage Order use case:

1. In the Windows Explorer, open the **`Samples\Ordersystem`** folder in your Rational Rose installation folder.
2. Drag the Microsoft Word document **`Manageorder.doc`** from the Windows Explorer and drop it on the Manage Order use case in the browser in Rational Rose. Rational Rose connects the file to the use case.
3. The **`Manageorder.doc`** file contains the flow of events description of the Manage Order use case. Open the document by double-clicking on the file in the browser, and take a look at its contents.
4. Exit Microsoft Word.

Viewing the Scenarios

Manage Order is the use case you performed in the previous (Entering a New Order) exercise. To understand how the design objects interact to perform the use case, look at one of its sequence diagrams.

1. In the browser you can see that there are two sequence diagrams that describe the Manage Order scenario. Create and Register Order describes the main flow, and Create Order Rows is a subflow that describes the details about how an order row is created. Open the Create and Register Order sequence diagram by double-clicking on it.

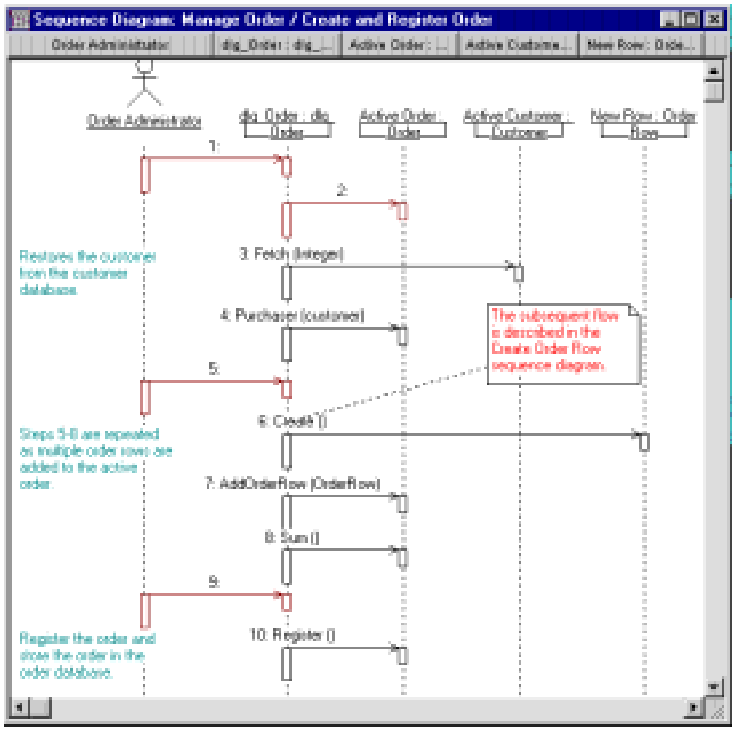

Sequence Diagram: Manage Order / Create and Register Order
Order Administrator
dlg_Order : dlg_...
Active Order: ...
Active Custome...
New Row: Orde...
Order Administrator
dlg_Order : dlg_Order
Active Order : Order
Active Customer : Customer
New Row : Order Row
1:
2:
3: Fetch (Integer)
4: Purchaser (customer)
5:
6: Create ()
7: AddOrderRow (OrderRow)
8: Sum ()
9:
10: Register ()
Restores the customer from the customer database.
The subsequent flow is described in the Create Order Row sequence diagram.
Steps 5-8 are repeated as multiple order rows are added to the active order.
Register the order and store the order in the order database.

Figure 5 Create and Register Order Sequence Diagram

2. Take a look at the contents of the sequence diagram:
   - The actor, Order Administrator, represents a person whose responsibility is to register orders.
   - Because a sequence diagram describes a specific scenario (instance) of a use case, the columns represent the objects (and not classes) that participate in the scenario. Active Order, for example, is an instance of the Order class.
   - Steps 1-10 in the scenario involve the passing of messages between the actor (Order Administrator), the user interface (dlg_Order), and the participating business objects (Active Order, Active Customer, and New Row).

Mapping the Scenarios to Visual Basic Code

After browsing the sequence diagram, you should view how the mechanisms described in the scenarios are modularized and implemented in the Visual Basic project components.

Message 1

First you will take a look at the source code for the Visual Basic form that corresponds to the dlg_Order object:

1. Make sure that Microsoft Visual Basic is running and that the **`ordersys.vbp`** project file is loaded.
2. In Rational Rose, select the dlg_Order object in the sequence diagram. To find the corresponding class in the model, click **Browse > Referenced Item**. This opens a class diagram that contains the class you are looking for.
3. Right-click on the dlg_Order class in the opened diagram and click **Browse Source** on the displayed menu. This opens the dlg_Order form specification in Microsoft Visual Basic.
4. First, take a look at what initially happens when a new order object is created (message 1). The general declarations section of dlg_Order says:

   ```
   Private Active_Order As New Order
   Private Active_Customer As New Customer
   Private My_Customers As New Customers
   ```
5. Close the class diagram with dlg_Order.

This code creates three objects when the dlg_Order form is loaded. As you can see, Active_Order corresponds to the Active Order object in the sequence diagram, and Active_Customer corresponds to the Active Customer object.

Message 2

The constructor in the Order object will allocate a new order ID at the object creation time (message 2). To view the code for that method:

1. In Rational Rose, select the Active Order object in the sequence diagram and click **Browse > Referenced Item**.
2. Right-click on the Order class in the opened diagram and click **Browse Source** on the displayed menu. This opens the Order class module specification in Microsoft Visual Basic.
3. Scroll down to the Class_Initialize method:

```
Private Sub Class_Initialize()
    On Error GoTo Order_Class_Initialize__exception
    ’allocate the object attributes
    Set Orderrows = New Collection
    ’set the order id to something
    mOrder_Id = 123
    Exit Sub
Order_Class_Initialize__exception:
    Resume Order_Class_Initialize__end
Order_Class_Initialize__end:
    Exit Sub
End Sub
```

As you can see, a clear separation of concerns is maintained between the user services and the business objects. The Order business object is responsible for maintaining its properties while the user service, dlg_Order, is responsible for the presentation of the properties.

Message 5

Now, take a look at the creation and editing of a new order row (message 5):

1. In Microsoft Visual Basic, select dlg_Order and click **View > Object**.
2. Double-click **New** in the order rows pane. This brings up the source code for the btn_new_Click() event handler:

```
Private Sub btn_new_Click()
    Dim New_Row As New OrderRow
    'Edit the new row.
    If New_Row.Create() Then
        'Ok clicked!
        'Add the new row to the active order
        Active_Order.Add_OrderRow New_Row
        'Add the new row to the dialog
        lst_order_rows.AddItem _
             New_Row.Quantity & " units of " & _
             New_Row.The_Article.Name & " @ " & _
               Format(New_Row.The_Article.Price, "$#.#0")
        'Calculate and present the total order sum.
        txt_sum.Text = Format(Active_Order.Sum(), "$#.#0")
    End If
End Sub
```

This is another example of the separation between user services and business services, because the responsibility for creation and initialization of an OrderRow object (the Create method) has been assigned to the object itself. Also, the Sum method has been assigned to the Order object.

Finally, return to Rational Rose and close the opened class diagram, sequence diagram, and use-case diagram.

Chapter 3

Browsing the Order System Design Model

In this exercise, you will use Rational Rose to get acquainted with the design model of the order system. You will also see how the model is prepared for code generation.

Rational Rose provides you with several mechanisms for browsing the model. The tutorial takes you through the following exercises:

1. Opening the order system design model
2. Browsing the model
3. Browsing the class diagrams
4. Browsing the specifications
5. Examining how the model is prepared for code generation

Opening the Order System Design Model

Make sure that Rational Rose is running and that the order system model is open.

1. Expand the Logical View in the browser by clicking the “**+**” sign.
2. Open the Three-Tiered Service Model diagram by double-clicking on its name in the browser. This diagram shows the architecture of the order system application, that is, the classes of the design model and the relations between them. The diagram notation is a subset of the Unified Modeling Language (UML).

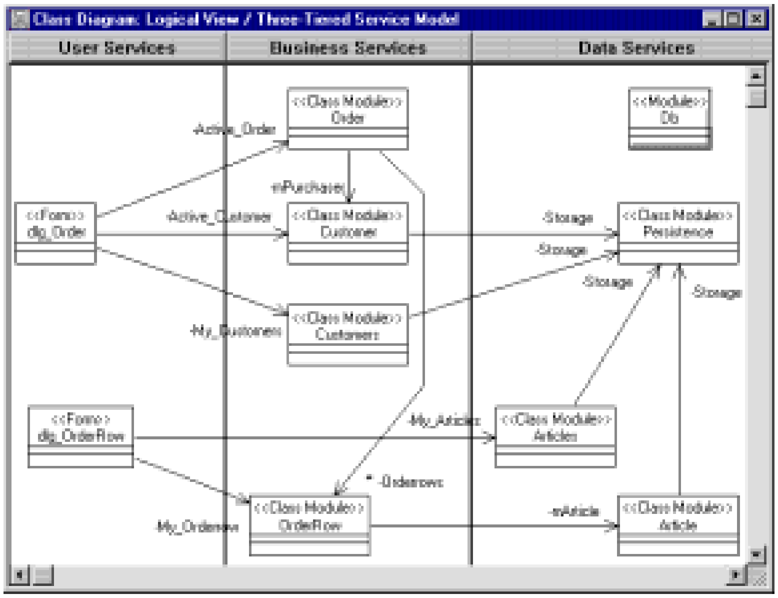


Figure 6 Three-Tiered Service Model Diagram

The Three-Tiered Service Model diagram supports the three-tiered architectural approach used when building Microsoft Visual Basic applications, as it separates the components of the system into three conceptual tiers of services: User Services, Business Services, and Data Services.

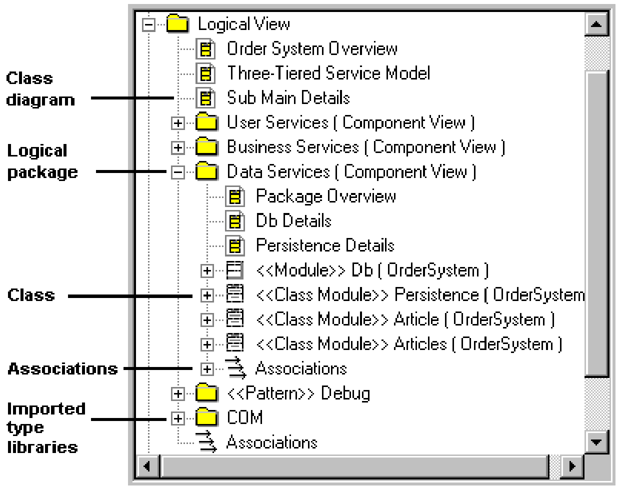
Note: *To be able to create new three-tiered diagrams, the* **3 Tier Diagram** *option on the* **Diagram** *tab of the* **Options** *dialog box must be selected. You do not have to change that option for this tutorial, but if you change it you must restart Rational Rose for the change to take place.*

Browsing the Model

As you can see in the browser, a system can be described from four different views:

- The use-case view, which you examined in the previous exercises.
- The logical view is the actual design model of the system, which you will examine and change during the next exercises.
- The component view describes the physical structure of the system and the mapping between the classes in the model and software components.
- The deployment view, which is empty in this example, is used to show the different processes of the running system and how they are related.

1. Expand the Data Services package in the logical view. The different symbols in the browser mean:

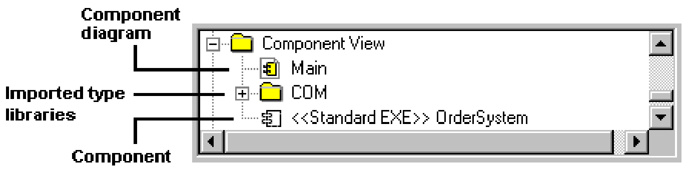


Logical View
Order System Overview
Three-Tiered Service Model
Sub Main Details
User Services (Component View)
Business Services (Component View)
Data Services (Component View)
Package Overview
Db Details
Persistence Details
<<Module>> Db (OrderSystem)
<<Class Module>> Persistence (OrderSystem
<<Class Module>> Article (OrderSystem)
<<Class Module>> Articles (OrderSystem)
Associations
<<Pattern>> Debug
COM
Associations

Figure 7   Expanded Logical View in the Browser

The “<<...>>” in front of the name of a model element indicates the stereotype of the element. Rational Rose uses the stereotype of a model element during code generation to determine what kind of Visual Basic item to generate from it.

2. Expand the Component View and take a look at its contents. The different symbols in the browser mean:



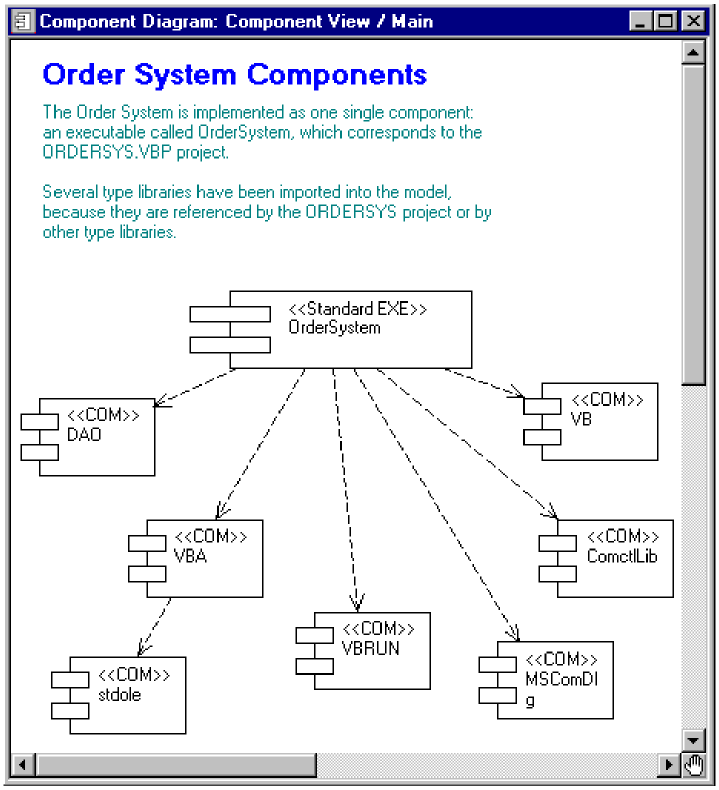
Component View
Main
COM
<<Standard EXE>> OrderSystem

Figure 8 Expanded Component View in the Browser

Note that the order system is implemented as one single component, an executable called OrderSystem, which corresponds to the **`ordersys.vbp`** project. All classes in the model are assigned to, and thereby implemented by, the OrderSystem component.

The COM package contains existing COM components that are referenced from the **`ordersys.vbp`** project. The type libraries of those components have been imported into the model. There is a logical package for each such type library in the Logical View as you can see in Figure 7.

3. Open the Main diagram of the Component View, which uses dependency relationships to show how the components are related.


Component Diagram: Component View / Main

Order System Components

The Order System is implemented as one single component: an executable called OrderSystem, which corresponds to the ORDERSYS.VBP project.

Several type libraries have been imported into the model, because they are referenced by the ORDERSYS project or by other type libraries.

Figure 9 Main Component Diagram

4. Close the Main component diagram.

Browsing the Class Diagrams

As you have seen already, the design model is illustrated in several diagrams, each with its own purpose. The Three-Tiered Service Model diagram, located on the top level in the Logical View, is the main diagram of the order system design model. There are other diagrams that illustrate interesting details about parts of the model.

For example, look at the details about the Order class:

1. Double-click on the Order Details diagram in the Business Services package in the browser. This diagram shows the details about the Order class, such as its methods, its properties, and its relations with other classes.

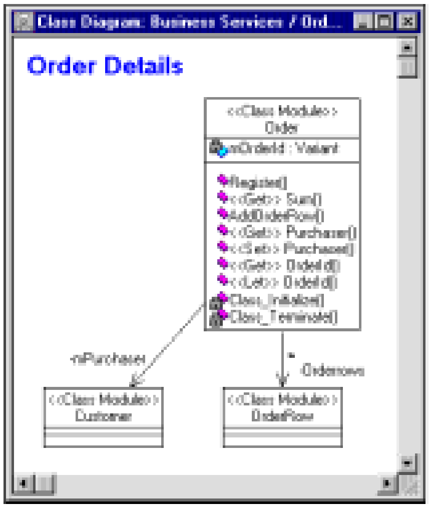


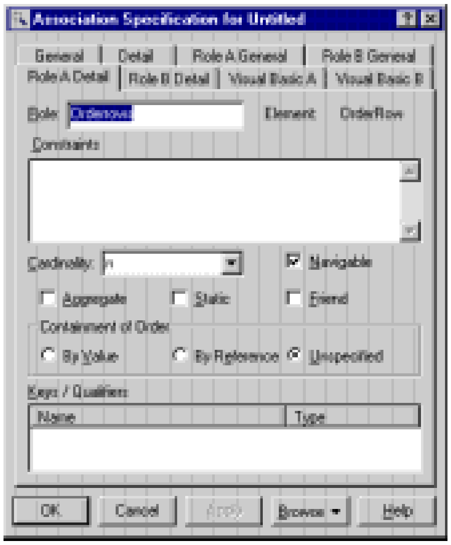
Figure 10 Order Details Diagram

2. The Orderrows relationship is marked with a “*” sign. This indicates that the Order class may reference many OrderRow objects. Also, note that property procedures are assigned the stereotypes Get, Set, and Let in the model.
3. A class may occur in several diagrams. To get a list of all the diagrams where the Order class is used, select the class and click **Report > Show Usage**.
4. Select one of the diagrams in the list and click **Browse**. Close the **Show Usage** dialog box.
5. Look at the diagram and the relationships between the classes.
6. Close the opened diagram and the Order Details diagram.

Browsing the Specifications

Each model element is defined by a specification, which holds all model information about that element. To view the specification of the relationship between the Order and OrderRow classes:

1. Double-click on the relationship between the Order and OrderRow classes in the Three-Tiered Service Model diagram. The specification of that relationship opens.
2. Examine the information behind the different tabs. Note that an association relationship consists of two roles, A and B; one role on each end of the relationship. A role indicates the role that a class on one end plays against the other class.
3. Click the **Role A Detail** tab. The role name Orderrows indicates that the OrderRow class constitute the order rows for the Order class. Also, note that the **Navigable** option is selected. Rational Rose generates Visual Basic code only for those roles that are navigable.



Association Specification for Untitled

General | Detail | Role A General | Role B General
Role A Detail | Role B Detail | Visual Basic A | Visual Basic B

Role: Orderrows   Element: OrderRow

Constraints:

Cardinality: n   [✓] Navigable

[ ] Aggregate   [ ] Static   [ ] Friend

Containment of Order: ( ) By Value   ( ) By Reference   (●) Unspecified

Keys / Qualifiers:

Name	Type

Figure 11   Association Specification—Role A Detail Tab

4. As you can see, each role has a **Visual Basic** tab. The options on this tab are called model properties. They are used during code generation to customize the mapping of the model element to Visual Basic code. There is a separate tool, called the Model

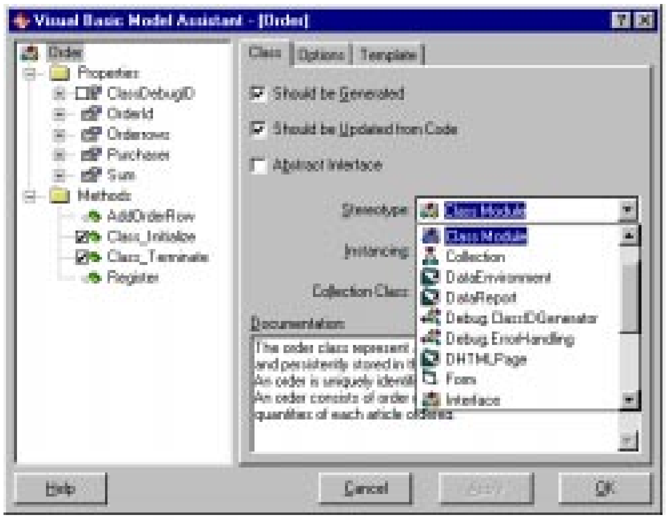
Assistant, for updating the model properties and for previewing the code to be generated for each class and member. You will use the Model Assistant in the next exercise.

5. Close the specification.

Examining How the Model Is Prepared for Code Generation

The next step is to study how the model is prepared for code generation. Most of the implementation details are specified in the class, relationship and method specifications, but some implementation details must be entered as stereotypes or model properties of the model elements. The Model Assistant tool is used to specify Visual Basic classes.

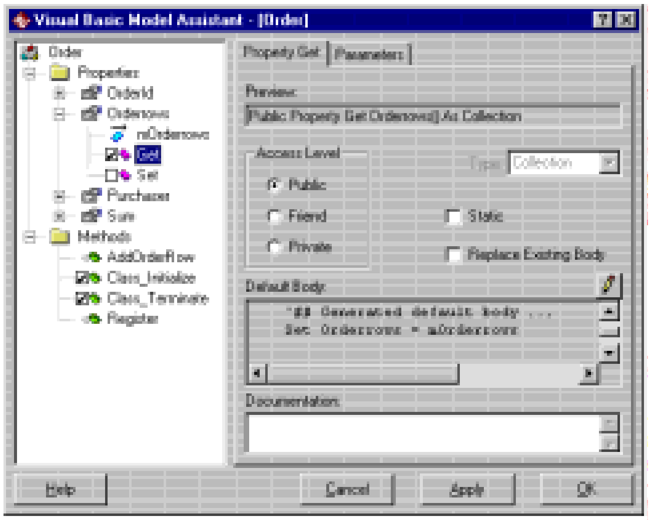
1. Open the Model Assistant for the Order class by right-clicking on the class and clicking **Model Assistant** on the displayed menu.
2. As you can see on the **Class** tab, the stereotype is set to “Class Module”. When generating code for a class, Rational Rose creates a Visual Basic item of the specified type. Thus, the Order class has been generated as a class module in the corresponding Visual Basic project. To get a list of other available types of Visual Basic classes, including any user-defined stereotypes, click the arrow in the **Stereotype** box.



The order class represent
and persistently stored in
An order is uniquely identi
An order consists of order
quantities of each article ordered.

Figure 12 Model Assistant—Order Class

3. The left list shows the properties and methods that will be generated into the Order class module. Under the Properties folder you can find both the class' properties and its roles in the model, because they all become properties in the code.
4. Expand Orderrows, which represents a placeholder for the corresponding role and any associated property procedures. As you can see, there are no property procedures associated with this role.
5. Select the check box next to Get to associate a Property Get procedure with the Orderrows role. Note that the Model Assistant automatically adds an "m" to the name of the role, in order to avoid a name collision with the new Property Get procedure. In the **Preview** box you can see the code that will be generated for the property procedure.



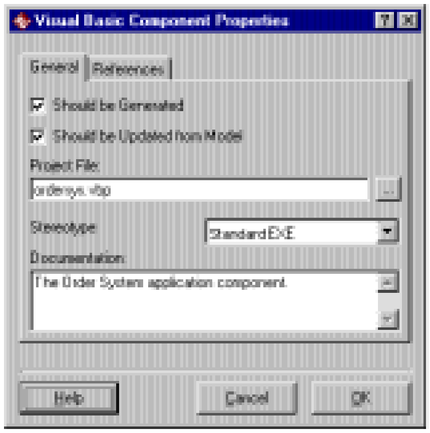
Preview:
```
Public Property Get Orderrows() As Collection
```

Default Body:
```
'## Generated default body ...
Set Orderrows = mOrderrows
```

Figure 13 Model Assistant—A Property Get Procedure

6. If you would click **Apply** or **OK**, the Property Get procedure is inserted into the model. Now, click **Cancel**.
7. To generate Visual Basic code for a class, the class must be assigned to a component with the Visual Basic language. The best way to view and change the assignment of classes to components in the model is to use the Component Assignment Tool. Thus, click **Tools > Visual Basic > Component Assignment Tool**.

8. Under Visual Basic, select the OrderSystem component, which represents the Visual Basic project that implements the order system. In the right list, you can see the classes that are currently assigned to that component.
9. Right-click on the OrderSystem component and click **Properties**. As you can see in the **Project File** box, this component corresponds to a Visual Basic project file named **`ordersys.vbp`**.


Visual Basic Component Properties

General | References

☑ Should be Generated

☑ Should be Updated from Model

Project File: ordersys.vbp

Stereotype: Standard EXE

Documentation: The Order System application component.

Figure 14 Component Properties Dialog Box

10. Close the **Visual Basic Component Properties** dialog box and the Component Assignment Tool.

Chapter 4

Round-Trip Engineering with Rational Rose Visual Basic

The Visual Basic project that implements the order system, **`ordersys.vbp`**, has been generated from the model that you have been browsing during the previous exercises. The code was then further refined in the Visual Basic environment and the changes were reverse engineered into the model. The process of alternating between the model and the code is called round-trip engineering.

In this part of the tutorial, you are going to create a new class in the model and generate it into the project by using the Code Update Tool. Finally, you are going to reverse engineer some code changes into the model, using the Model Update Tool.

The folder containing the order system model also contains the source code for the application. You use this code to practice round-trip engineering in the following exercises:

1. Creating classes and associations in the model
2. Updating the code from changes in the model
3. Browsing the code and the model
4. Updating the model from changes in the code

Creating Classes and Associations in the Model

Now, you will practice the round-trip possibilities by creating a new business service class and an association in Rational Rose, and then updating the code with those changes.

1. Open the Package Overview diagram for the Business Services package.
2. Create a new class by clicking the class symbol ( ) in the toolbox. Position the cursor at the desired location in the diagram and click to place the class in the diagram.
3. Assign the new class to the component that will implement this class by dragging the OrderSystem component from the Component View in the browser and dropping it on the new class in the Package Overview diagram. Rational Rose needs that information to know which Visual Basic project the class belongs to.
4. Create a relationship between the new class and Order by clicking the association symbol ( ) in the toolbox. Point to the new class in the diagram and drag the association to Order. Release the mouse button.
5. Name the new role of the Order class by right-clicking on the Order side of the association and selecting **Role Name** from the displayed menu. By default, the name of the new role is “theOrder”, as illustrated in Figure 15. The “+” means that the role is Public.

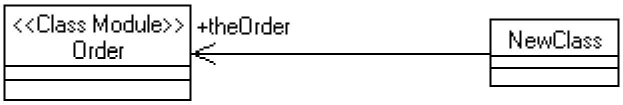
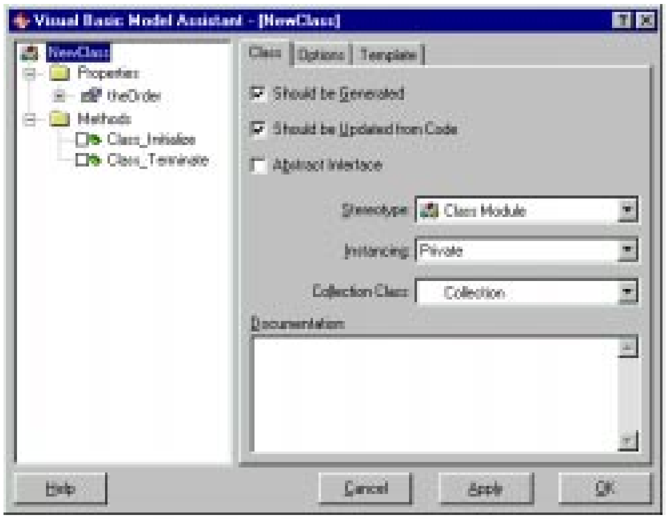


Figure 15 Association Relationship between the New Class and Order

6. To create methods and properties on the new class, right-click on the class and select **Model Assistant** from the displayed menu. Note that there is a property defined for the association that you just created.


Visual Basic Model Assistant - [NewClass]

NewClass
- Properties
  - theOrder
- Methods
  - Class_Initialize
  - Class_Terminate

Class | Options | Template

- [✓] Should be Generated
- [✓] Should be Updated from Code
- [ ] Abstract Interface

Stereotype: Class Module
Instancing: Private
Collection Class: Collection
Documentation:

Figure 16 The Model Assistant for the New Class

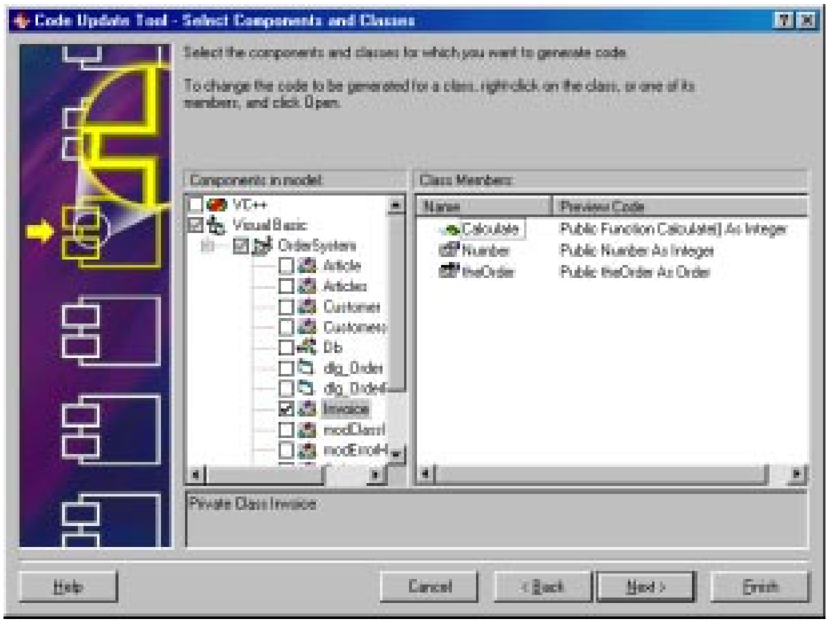
7. Give the new class a name by selecting NewClass in the left list and clicking on the name. Type **`Invoice`**.
8. Create a method by right-clicking on the Methods folder in the left list and selecting **New Method** from the displayed menu. Give the new method the name **`Calculate`**. Define the method's return type by selecting **Integer** in the **Type** box.
9. Create a property by right-clicking on the Properties folder in the left list and selecting **New Property** from the displayed menu. Give the new property the name **`Number`**. Define its type by selecting **Integer** in the **Type** box. Click **OK**.

Now you have created a new class with a property, a method, and a relationship.

Updating the Code from Changes in the Model

When you are satisfied with the changes in the model, it is time to generate code for the new Invoice class.

1. Right-click on the Invoice class and click **Update Code** on the displayed menu. The Code Update Tool starts. Click **Next** if the **Welcome** page is shown.
2. Expand the OrderSystem component by clicking the "+" sign next to it, and make sure that the Invoice class is selected. To display a preview of the code to be generated from Invoice, click on the class:


Code Update Tool - Select Components and Classes

Select the components and classes for which you want to generate code.

To change the code to be generated for a class, right-click on the class, or one of its members, and click Open.

Components in model:
- [ ] VC++
- [✓] Visual Basic
  - [✓] OrderSystem
    - [ ] Article
    - [ ] Articles
    - [ ] Customer
    - [ ] Customers
    - [ ] Db
    - [ ] dlg_Order
    - [ ] dlg_OrderI
    - [✓] Invoice
    - [ ] modClassI
    - [ ] modErrorH

Class Members:

Name	Preview Code
Calculate	Public Function Calculate() As Integer
Number	Public Number As Integer
theOrder	Public theOrder As Order

```
Private Class Invoice
```

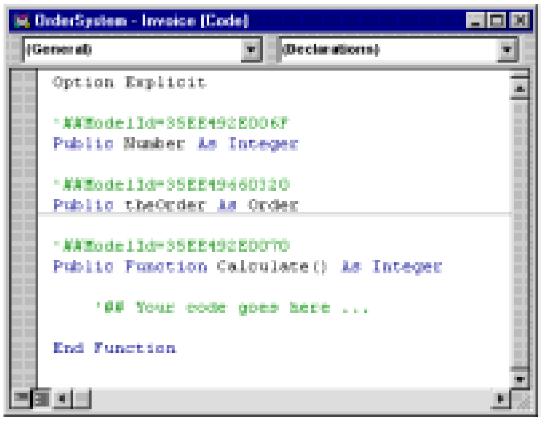
Figure 17 Code Update Tool

3. Click **Next**.
4. The **Finish** page is shown, where you can see a summary of what will be generated. Now, click **Finish** to start the code generation.
5. If the **Save Model As** dialog box appears, click **Cancel**.
6. When the code generation is completed a **Summary** page is shown. Take a look at the result on the **Summary** and **Log** tabs, and then click **Close**.

Browsing the Code and the Model

The Rational Rose and Microsoft Visual Basic tools are tightly integrated, not only with respect to code and class generation, but you can also use each application to browse the corresponding code or components in the other application.

1. To view the class module that was generated from the Invoice class, right-click on the Invoice class in a diagram and click **Browse Source**.
2. As you can see in the displayed Microsoft Visual Basic application window, a class module called Invoice has been created in the project. Note how the property, method, and relationship were mapped to Visual Basic code. The generated code depends on the code update settings in the **Visual Basic Properties** dialog box in Rational Rose. In Figure 18, the **Generate debug code** and **Generate error handling** code options were both cleared.



```
Option Explicit

'##ModelId=35EE492E006F
Public Number As Integer

'##ModelId=35EE49660320
Public theOrder As Order

'##ModelId=35EE492E0070
Public Function Calculate() As Integer

    '## Your code goes here ...

End Function
```

Figure 18 Code Generated from the Invoice Class

Note: *For each generated project item, member, and method, the code generator adds an identifier as a code annotation (for example “*`ModelId=35EE49660320`*”), which identifies the corresponding class, property, role, or method in the model. Do not edit or remove those identifiers!*

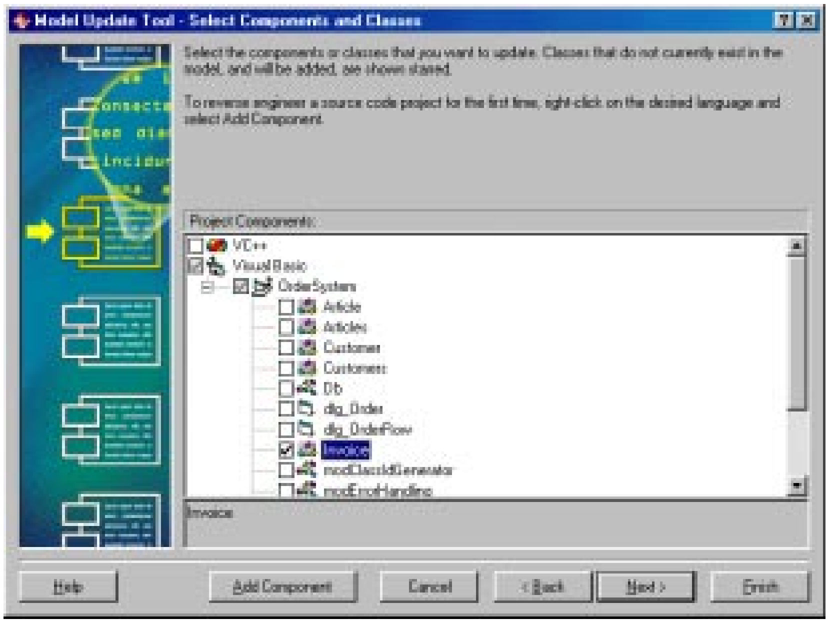
3. To browse from the code to the model, click **Add-Ins > Rational Rose 2000** in Microsoft Visual Basic. The Rational Rose tool window opens.

4. Right-click on ordersys.mdl in the Rational Rose tool window, and click **Browse Model**. Rational Rose is displayed (and started if needed) with the model opened.

Updating the Model from Changes in the Code

Now, suppose that you need to make some changes in the Invoice class module, and want to update the design model with those changes.

1. In Microsoft Visual Basic, change the name of Invoice’s Number property to Sum.
2. In the Rational Rose tool window, right-click on ordersys.mdl and click **Update Model**. The Model Update Tool starts. Click **Next** if the **Welcome** page is shown.
3. Expand the OrderSystem component by clicking the "+" sign next to it, and make sure that only the Invoice class is selected.


Model Update Tool - Select Components and Classes

Select the components or classes that you want to update. Classes that do not currently exist in the model, and will be added, are shown shaded.

To reverse engineer a source code project for the first time, right-click on the desired language and select Add Component.

Project Components:

- [ ] VC++
- [✓] Visual Basic
  - [✓] OrderSystem
    - [ ] Article
    - [ ] Articles
    - [ ] Customer
    - [ ] Customers
    - [ ] Db
    - [ ] dlg_Order
    - [ ] dlg_OrderRow
    - [✓] Invoice
    - [ ] modClassIdGenerator
    - [ ] modErrorHandling

Invoice

Figure 19 Model Update Tool

4. Click **Next** and take a look at the summary of what will be updated in the model.

5. Click **Finish** to update the model.
6. If the **Save Model As** dialog box appears, click **Cancel**.
7. When the Model Update Tool is finished, the result is shown on a **Summary** page. The **Summary** tab shows the effects of the reverse engineering, and the **Log** tab lists any warnings or errors.
8. Click **Close**.
9. In Rational Rose, note that the name of the property has changed also in the model.

You have now completed the basic steps in using Rational Rose for Visual Basic. For complete information about Rational Rose, please refer to the Rational Rose manuals and online help.