Getting Started Guide

for Borland® Together® Edition for Microsoft® Visual Studio® .NET

(version 2.0)

Borland® Together®

Integrated and Agile Design Solutions

Borland®
Excellence Endures™

Borland Software Corporation
100 Enterprise Way
Scotts Valley, California 95066-3249
www.borland.com

TogetherSoft Corporation, a wholly owned subsidiary of Borland Software Corporation, may have patents and/or pending patent applications covering subject matter in this document. Please refer to the product CD or the About dialog box for the list of applicable patents. The furnishing of this document does not give you any license to these patents.

Copyright © 2003-2004 TogetherSoft Corporation, a wholly owned subsidiary of Borland Software Corporation. All rights reserved. All Borland brand product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Microsoft and Visual Studio are registered trademarks of Microsoft Corp. in the United States and/or other countries. All other marks are the property of their respective owners.

Updated: May 20, 2004

Contents

CHAPTER

1

Introducing Borland Together Edition for Microsoft Visual Studio .NET

This chapter includes the following topics:

About Borland Together Edition for Microsoft Visual Studio .NET

Borland Together Edition for Microsoft Visual Studio .NET (Together VS .NET) is a UML modeling extension package for Microsoft® Visual Studio® .NET 2003 (Visual Studio .NET). Together VS .NET provides a new edition to the Borland product line of the award-winning, design-driven environment with features such as UML modeling, patterns, QA audits, code refactoring, XMI import and export, and documentation generation combined seamlessly with your C# and Visual Basic .NET projects in the Visual Studio .NET development environment.

Note QA audits and code refactorings are available only for C# projects.

A key feature of Together VS .NET, LiveSource™, keeps your Together VS .NET diagrams synchronized with your source code in the Visual Studio .NET editor.

This guide assumes that you have installed Together VS .NET and Visual Studio .NET on your computer. You can have Together VS .NET and Visual Studio .NET installed in any location. There is no advantage or disadvantage to coordinating installations or installing on the same root directory. For information about installing Together VS .NET or Visual Studio .NET, refer to the respective product documentation.

About this guide

The goal of this guide is to provide you with a basis for understanding and using Together VS .NET. It explains how to create a sample C# project and then uses the project to exercise the primary features of Together VS .NET.

As such, it is recommended that you follow the examples in this guide in sequential order. For more task-oriented instructions, refer to the online help for Together VS .NET. Choose **Contents** from the Visual Studio .NET Help menu, and select, *Borland Together Edition for Microsoft Visual Studio .NET*, in the Contents pane.

Note The instructions provided in this guide are specific to Borland Together Edition for Microsoft Visual Studio .NET.

This guide provides examples for:

- "Navigating around Together VS .NET"
- "Creating a project and opening the Model and Diagram Views"
- "Creating a use case diagram"
- "Using a class diagram"
- "Creating a sequence diagram"
- "Using the Model View"
- "Using the Properties Window"
- "Using the Overview"
- "Managing diagram views"
- "Using and creating patterns"
- "Running quality assurance"
- "Refactoring"
- "Generating documentation"
- "Exporting and Importing XMI projects"

Together VS .NET documentation set

The documentation set for Together VS .NET consists of the items listed in Table 1.

Table 1 Together Edition for Microsoft Visual Studio .NET documentation

Item	Description	Location
Readme	Late-breaking information including: • Licensing notes • System requirements • Installing and starting Together VS .NET • Known problems	Readme located as follows: • The Docs directory of your Together VS .NET installation: `Docs/readme.html` • http://info.borland.com/techpubs/together
What's New	Information detailing new and improved features in Together Edition for Microsoft Visual Studio .NET	What's New located as follows: • The Docs directory of your Together VS .NET installation: `Docs/what's_new.html` • http://info.borland.com/techpubs/together
Getting Started Guide	Information for the first time user including: • Creating a C# project for use in Together VS .NET and Visual Studio .NET • Working with use case and class diagrams • Setting options in Together VS .NET • Other information specific to using Together VS .NET	PDF located as follows: • The Docs directory of your Together VS .NET installation: `Docs/gettingStarted.pdf` • http://info.borland.com/techpubs/together
Online Help	Online help for Together VS .NET. Comprehensive information most relevant to the user, including: • Introduction to Together VS .NET • Setting personal preferences and options • Detailed instructions for using Together VS .NET features • Working with UML diagrams in Together VS .NET	The Visual Studio .NET main menu under **Help | Contents**
Registering Together VS .NET	Setting up licensing for Together VS .NET	• The `Docs` directory of your Together VS .NET installation: `Docs/setting_up_licensing.html` • http://info.borland.com/techpubs/together

About licensing

Together VS .NET and Visual Studio .NET require separate licenses. For information regarding licensing for Visual Studio .NET, refer to the Visual Studio .NET documentation.

For information about Together VS .NET licensing, refer to the *Licensing and Registering Borland Together Edition for Microsoft Visual Studio .NET* (setting_up_licensing.html) document, included in the `Docs` directory of your Together VS .NET installation.

CHAPTER

2

Getting Started with Modeling

This chapter explains the basics of using Together VS .NET, including: how to navigate the Together VS .NET environment, how to create a project, and how to create diagrams.

This chapter includes the following topics:

Navigating around Together VS .NET

Together VS .NET provides a Model View and corresponding Diagram View and Properties Window to Visual Studio .NET. Combined, these tools are central to designing projects in UML and creating source code for your project. Moreover, Together VS .NET simultaneously synchronizes your diagrams and source code.

Together VS .NET, is integrated into your Visual Studio .NET development environment. The Together VS .NET modeling features are automatically activated for C# and Visual Basic .NET projects.

Note This guide models a C# project.

Together VS .NET brings three main components to the Visual Studio .NET development environment:

- **Model View** – The primary navigational view for Together VS .NET. You can use this view to manage diagram elements, open, create, delete, and so on. Explore the context menus of the different elements as you encounter them.
- **Diagram View** – Displays UML diagrams. This view provides a tab for each open diagram.
- **Properties Window** – Using Together VS .NET, this view shows the properties for an element selected from the Diagram or Model Views. Properties for each element are usually broken into two categories:
  - Design – Define foreground and background colors for design elements
  - General – UML properties for the selected element as well as source code properties for source code elements

Note For classifiers – classes, interfaces, delegates, enumerations, structures (C# projects only), and modules (Visual Basic .NET projects only) – the *Comments* category is also added to the Properties Window enabling you to add *Author*, *Since*, and *Version* comment tags to your source code.

Opening the Model and Diagram Views

To begin using Together VS .NET for an existing or new C# project, open the Model View and the corresponding Diagram view for your project.

To open the Model View and show the corresponding Diagram view for a C# project:

1. From the View menu, choose **Together VS .NET Model View**. The Model View opens.
2. Expand the root project node.
3. To open the Diagram View for the corresponding project, double click the *default* class diagram node  .

Setting global modeling options

Together VS .NET options allow you to globally deactivate Together VS .NET modeling support for all of your open C# and Visual Basic .NET projects or for all new C# and Visual Basic .NET projects using the Options dialog.

To deactivate modeling support for all new projects:

1. From the Tools menu, choose **Options**. The Options dialog opens.

2. Click the *Together VS .NET* folder to view Together VS .NET-specific configuration options. The General options display.
3. In the Options dialog, click on the *Automatically enable Together VS .NET support for new projects* field. A drop down arrow appears.
4. Click the drop down arrow. Select *False* from the drop down list, and click **OK**. This disables Together VS .NET modeling support. Be sure to set the value as *True* to complete this tutorial.

Tip You can also click on *Automatically enable Together VS .NET support for opened projects* to activate/deactivate modeling support for currently opened projects. This option works only for projects that have never been exposed to Together VS .NET support.

Creating a project and opening the Model and Diagram Views

This guide models a simple video store project to explore the various features of Together VS .NET.

To create a new project:

1. From the File menu, choose **New > Project**. The New Project dialog opens.
2. From the **Project Types** pane, choose *Visual C# Projects*.
3. From the **Templates** pane, choose *Empty Project*.
4. Enter “Video Store” as the project name.
5. Click **Ok**.

The project is created and displays in the Solution Explorer.

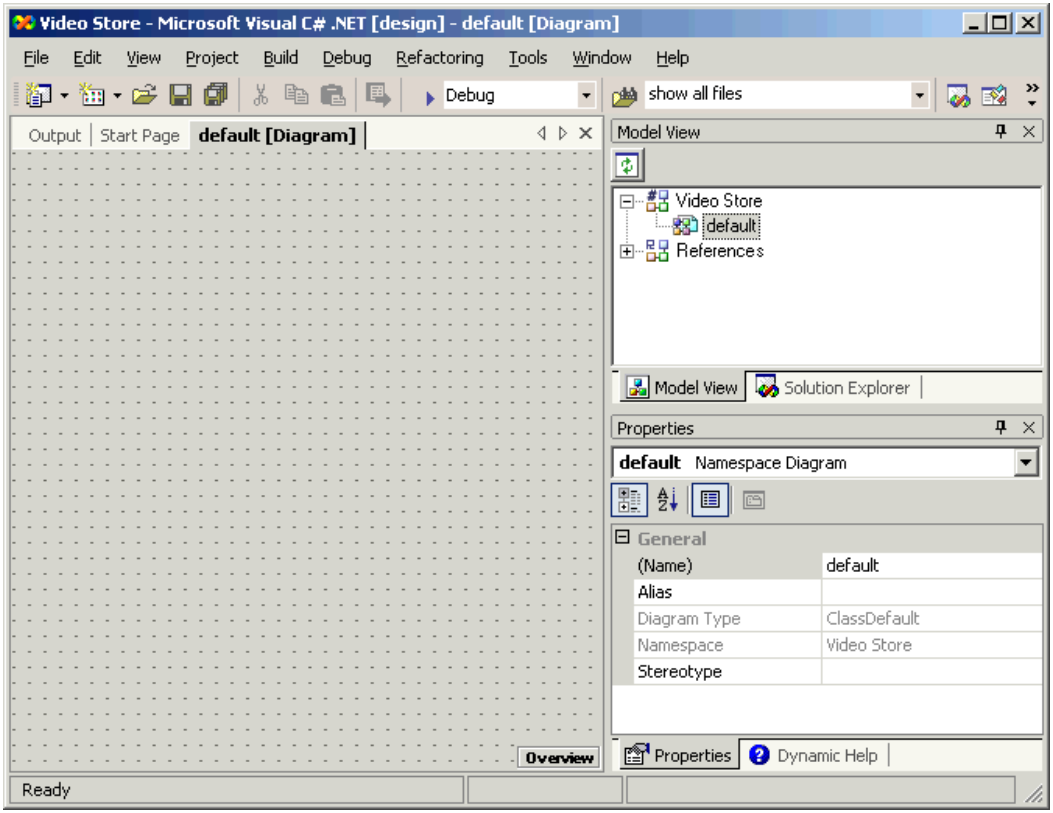
Tip To quickly open the Solution Explorer, enter **CTRL+ALT+L**.

To open the Model View:

1. From the View menu, choose **Together VS .NET Model View**. The Model View opens.
2. Expand the Video Store project node.
3. Double click the *default* class diagram node  . The Diagram View opens as shown in Figure 1.

Notice that the Model View is a dockable window. The Model View opens initially as a free-floating window; however, it can be free-floating or docked to a docking area. The docking areas for the Model View are comprised of any of the four borders of the Visual Studio .NET window. You can position the Model View window according to your preferences.

Figure 1 Video Store project as viewed using Together VS .NET



Creating a use case diagram

These instructions assume that you have created the Video Store project as explained in the previous section.

To create a use case diagram:

1. In the Model View, right click on the **Video Store** project node, and choose **Add > Other Diagram**.
2. In the resulting dialog box, choose **Use Case Diagram**. Enter “Video Store Use Case” as the name.
3. Click **OK**. The use case diagram opens in the Diagram View.

Tip You can use this dialog box to create any of the diagrams supported by Together VS .NET.

Populating the diagram

Use case diagrams are scenarios describing *what* your system does. These diagrams contain actors, use cases, and communication links as three main items of interest. The Toolbox and diagram context menu contain the design elements to populate the diagram.

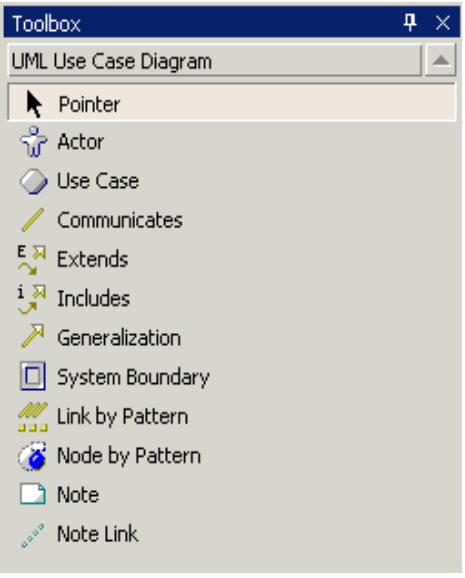
To populate the diagram:

1 Use the Toolbox to add design elements to the use case diagram. To open the Toolbox, choose **Toolbox** from the View menu.

2 To view the design elements, click **UML Use Case Diagram** on the Toolbox.

The UML Use Case Diagram Toolbox elements are shown in Figure 2.

Figure 2 Use Case Diagram Toolbox elements



3 Click the Actor button , and click on the diagram background.

4 The in-place editor automatically activates. Enter “Clerk” as the actor name. Press Enter to apply the name and close the in-place editor.

Tip To activate the in-place editor for diagram elements, double click the textual caption on the element.

Adding a system boundary

Add a system boundary to the diagram to separate the inventory system from the external actors.

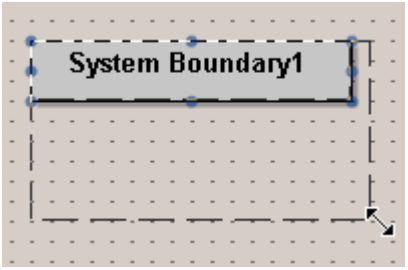
To add a system boundary:

1. Click the System Boundary button , and click the diagram background to the right of the Clerk.
2. Using the in-place editor, name the system boundary “Inventory System.”

Tip Alternatively, hold down the left mouse button, click the diagram background once, and drag the mouse over the diagram background to draw the System Boundary to a specific size.

To resize the system boundary, or any diagram element, select the system boundary object on the diagram. Drag the corner of the element to the desired size as shown in Figure 3. The element expands or contracts in the direction of the cursor.

Figure 3 Resizing the system boundary



Adding use case elements and communication links

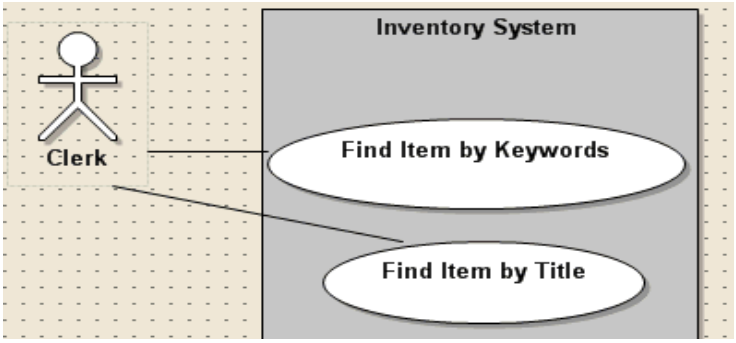
Use case elements represent a summary of scenarios for a single task, such as finding a movie by its title. Communication links show associations between actor and use case elements.

To add use case elements and communication links:

1. Click the Use Case button , and create two use cases inside the system boundary. Name the use cases “Find Item by Keywords” and “Find Item by Title.”
2. Click the Communicates button , and drag-and-drop from the actor to the use case. Repeat this step to create a link for each use case.

Your use case diagram should be similar to the diagram in Figure 4.

Figure 4 Sample use case diagram



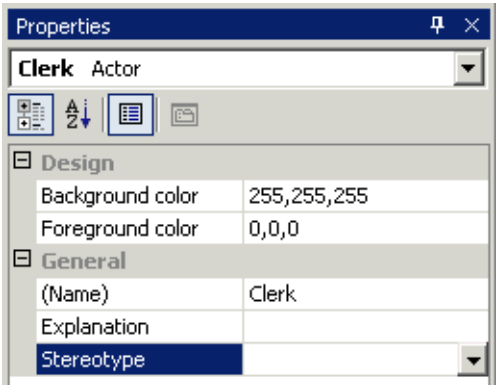
Adding a stereotype

Together VS .NET supports the use of stereotypes. You can adhere to UML-defined stereotypes or customize stereotypes based on your requirements.

To add a stereotype:

1 Select the **Clerk** actor to view the properties for Clerk in the Properties Window as shown in Figure 5. You can open the Properties Window by choosing **Properties Window** from the View menu. Alternatively, use the **F4** key on your keyboard.

Figure 5 Properties Window for Clerk



2 In the Properties Window, click on the *Stereotype* field. A drop down arrow displays as shown in Figure 5.

3 Click the drop down arrow, and select *manager* from the list.

Note Several properties contain the drop down arrow or the information button ... . These provide you with additional choices for selection or entry, either as drop down lists or dialogs.

Tip Once a stereotype has been added to a diagram element, you can make changes to the stereotype field using the in-place editor as shown in Figure 6.

Figure 6 Using the in-place editor on the stereotype field

Completing the scenario

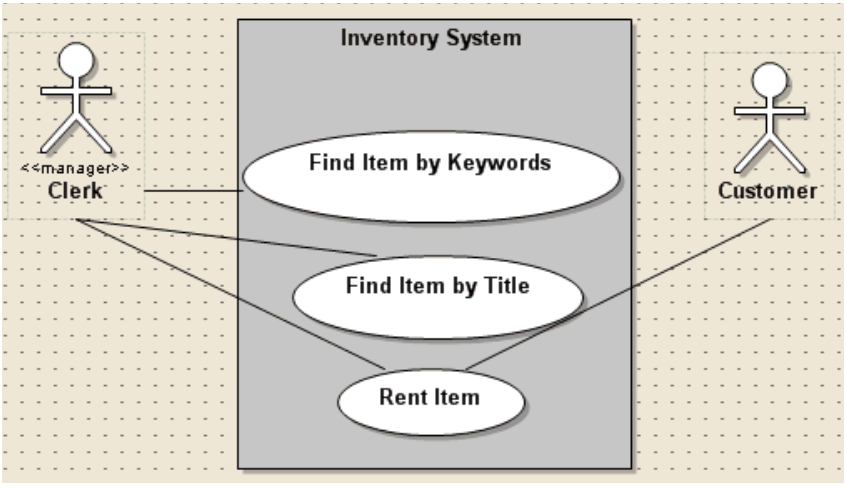
To complete the Video Store Use Case Diagram, we will need to add additional use case and actor elements and the communication links between the appropriate elements.

To complete the diagram:

1. Add another use case inside the *Inventory System* boundary. Name it “Rent Item.”
2. To the right of the Inventory System boundary, add another actor. Name it “Customer.”
3. Using the Communicates link, link the use case to both the Customer and Clerk actors.

Your diagram should be similar to the one shown in Figure 7.

Figure 7 Use case diagram for Video Store project



Using a class diagram

Together VS .NET creates a physical class diagram for each namespace in a project. The Video Store project has a default class diagram that represents the project at the root level.

To open and use the default class diagram:

1. In the Model View, expand the Video Store project node.
2. Double click the Video Store *default* class diagram node  . The default class diagram opens in the Diagram View.
3. Use the Toolbox to create a class. Click **UML Class Diagram** in the Toolbox to reveal the Class Diagram elements. Click the Class button  , and click on the diagram background. Using the in-place editor, rename the class "Store."
4. Select the **Store** class to view its details in the Properties Window.
5. Select the *Stereotype* field, click the drop down arrow, and select *place*.
6. Repeat steps 3-5 to create classes for `Clerk` (stereotype = role) and `Item` (stereotype = description).

As alternatives to the steps previously described, Together VS .NET provides two other methods for creating a class:

- Right click on the diagram background, and choose **Add > Class**.
- In the Model View, right click on the root project node or the default class diagram, and choose **Add > Class**.

Setting compartment controls

You can collapse or expand compartments for the different members of class and interface elements. By default, compartments are displayed on the diagram. You can use the Options dialog to set viewing preferences for compartment controls so that they are displayed as a straight line or with compartments. Compartment controls are particularly useful when you have large container elements with content that does not need to be visible at all times.

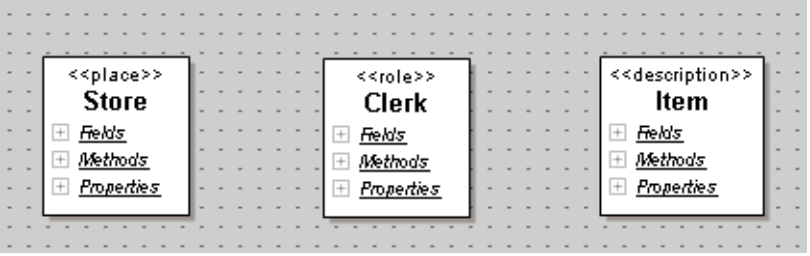
To view the compartment controls:

1. From the Tools menu, select **Options**. The Options dialog opens.
2. Click the **Together VS .NET** folder, and select the **Diagram** node.
3. Click on the *Show compartments as line* field.
4. Click the drop down arrow, and select *False.*
5. Click **OK**. Your diagram elements should be similar to Figure 8.

To collapse or expand compartments:

1. Select the class (or interface) on the diagram.
2. Click the “+” or “-” in the left corner of the compartment.

Figure 8 Video Store diagram with Store, Clerk, and Item classes



Adding methods and fields

As part of the requirements identified in the use case diagram for the Video Store Inventory system, Table 2 lists the methods and fields that you need to add to the classes you have created.

Table 2 Requirements for the Video Store sample project

Class	Fields	Methods
Store	address storeNumber hours phone	findByTitle findByKeywords findByItemNumber
Clerk	employeeId name	
Item	itemNumber title	getItemNumber setItemNumber getTitle setTitle

To create members for a class:

1. Open the Video Store class diagram, right click on the `Store` class and choose **Add > Field**. Alternatively, select the `Store` class on the diagram or in the Model View, and enter **Ctrl+W**.
2. The in-place editor activates. Using the in-place editor, name the field `address:string`. When you use the in-place editor to name a class member on the diagram, you can also enter its type.

Tip Alternatively, you can activate the in-place editor by selecting the field, and clicking **F2** on your keyboard.

3 In the Properties Window, click the *Visibility* field and choose *private* from the drop down list. By default, Together VS .NET creates *public int* fields and *public void* return methods.

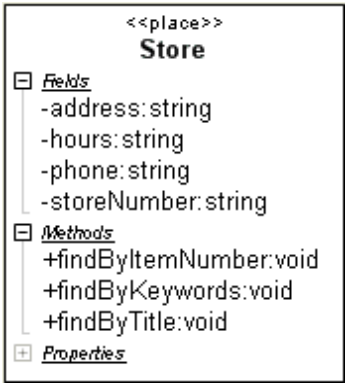
Tip Once you click the *Visibility* field, you can click the `Tab` key on your keyboard, and then use the down arrow on your keyboard to scroll through the different visibility values.

4 To add a method to the class, right click on the `Store` class and choose **Add > Method**. Alternatively, select the `Store` class on the diagram or in the Model View, and enter **Ctrl+M**.

5 Using the in-place editor, name the method `findByTitle`.

6 Repeat the previous steps to add the remaining fields and methods for the `Store` class as listed in Table 2. The Store class should be similar to Figure 9.

Figure 9 Store class with fields and methods



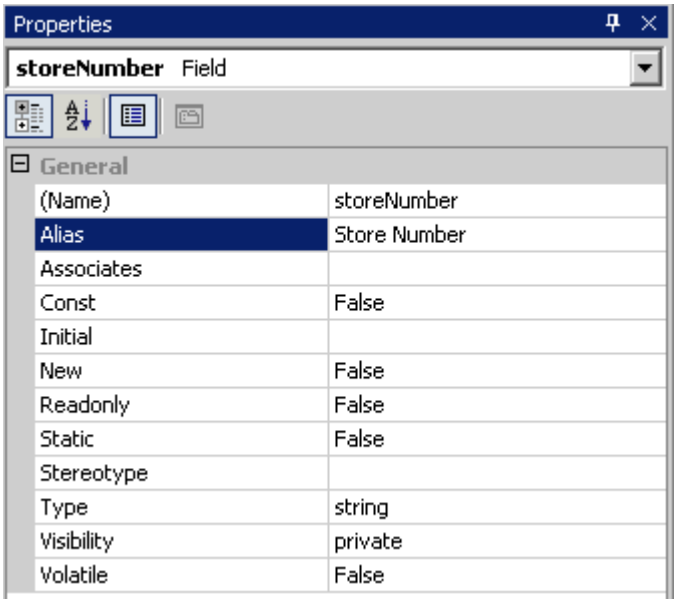
7 Continue adding the fields and methods listed in Table 2 to the `Clerk` and `Item` classes. For the `getItemNumber` and `getTitle` methods, specify *string* as the return type.

Tip Select a class member such as a field or method on the diagram or in the Model View, and use the **Insert** key on your keyboard to quickly add another member of the same type to the current class. After the new member has been added, press **F2** on your keyboard to activate the in-place editor. You can also use the shortcut keys displayed on the context menus to quickly add members and diagram elements.

8 Select the `storeNumber` field of `Store`. In the Properties Window, enter “Store Number” as the alias value as shown in Figure 10.

Tip The *alias* property is useful for specifying a different name from that in the actual code.

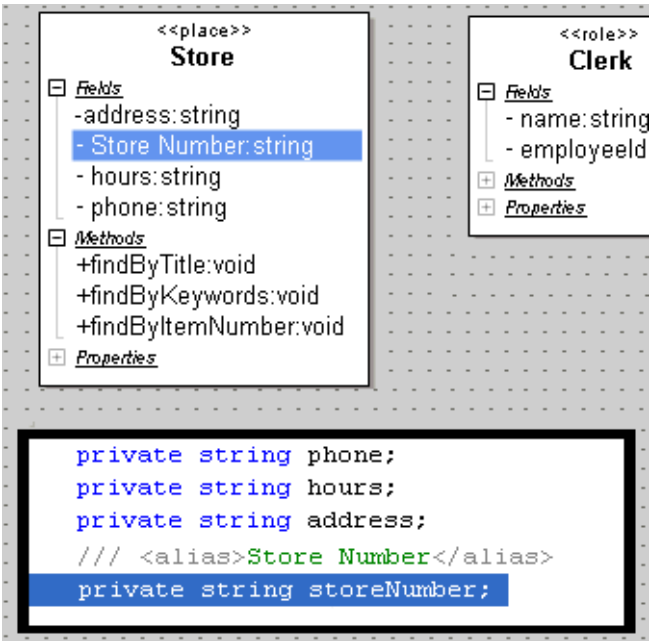
Figure 10 Creating an alias for a field



The alias changes only the visual label for the field in the diagram, as shown in Figure 11. The field remains `storeNumber` within the source code.

Tip To view the source code, double click the class node or right click on the field, and choose **Go to Definition**.

Figure 11 Alias for storeNumber and corresponding code



Creating relationships and links

The sample Video Store project includes two relationships:

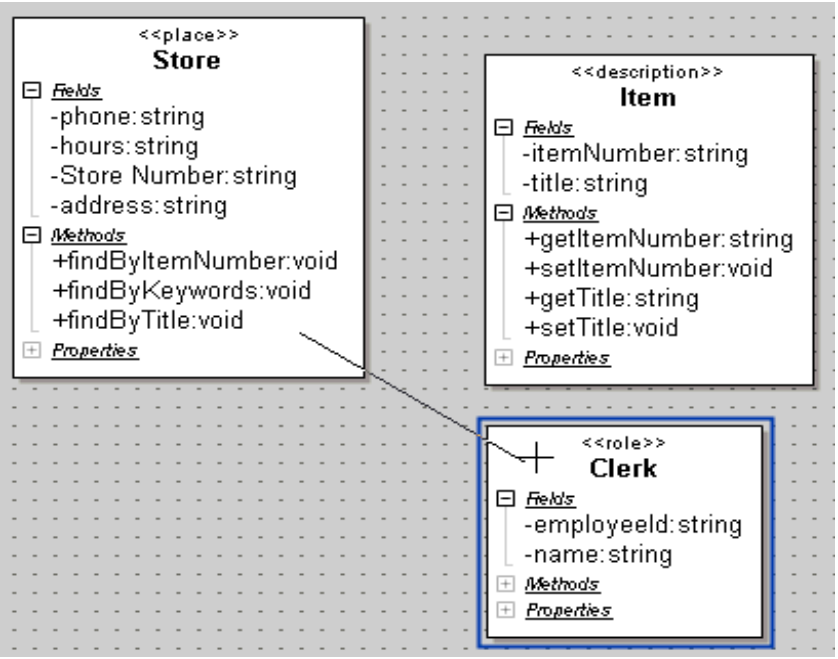
- A `Clerk` is associated with a `Store`
- A `Store` consists of many `Items`

You can create an association link to establish the relationship between `Clerk` and `Store` as “client” and “supplier.” For the second relationship, you can create an aggregation since `Store` has several `Items`. By using the **Link By Pattern** button, you can easily implement the aggregation as a collection.

To create relationships and links:

1 Click the **Association Link** button in the Toolbox. Click on `Store` and then drag-and-drop the link to `Clerk`. This action establishes the relationship from “client” to “supplier.” Notice that as you move over a class, a blue border highlights the location that Together VS .NET recognizes as a valid destination for dropping the end of the link as shown in Figure 12.

Figure 12 Using drag-and-drop to create an association link

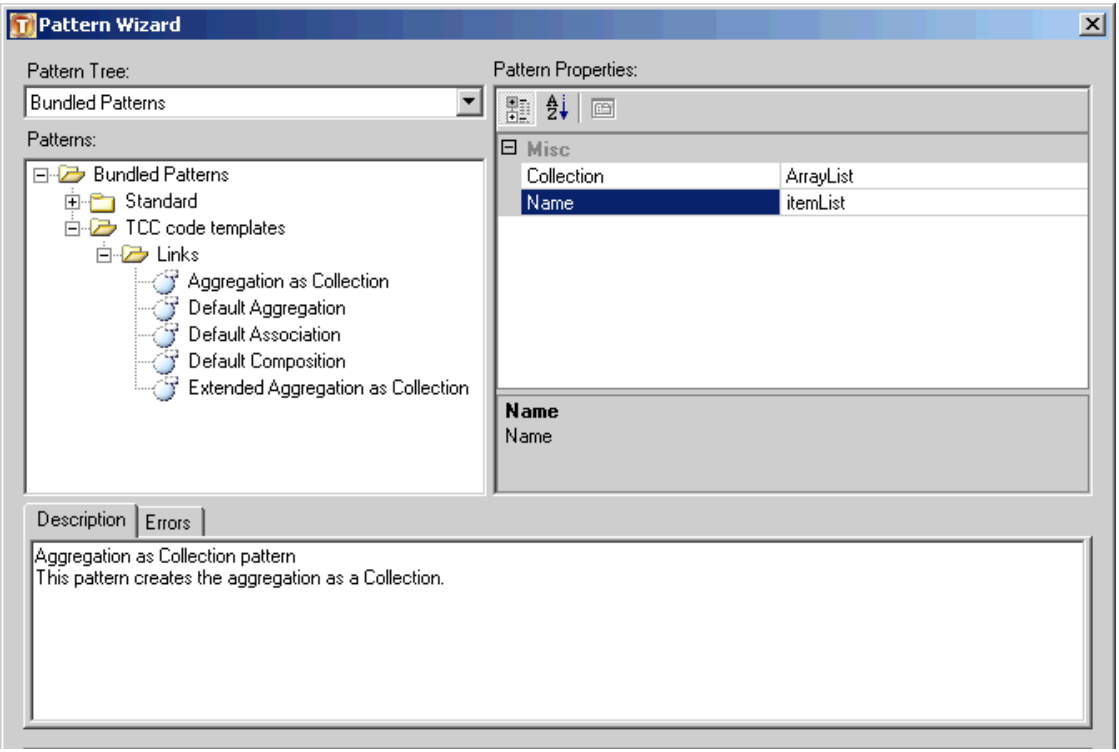


2 Click the **Link by Pattern** button in the Toolbox. Click on `Store` and then drag-and-drop the link to `Item`. The Pattern Wizard dialog opens.

3 In the Pattern Wizard, expand **TCC code templates > Links**, and select **Aggregation as Collection**.

4 As shown in Figure 13, for the field name enter “itemList” to indicate a collection, and click **Ok**.

Figure 13 Pattern Wizard dialog



Together VS .NET automatically adds the link information to the code by placing tags above the field. In this example, the link is an aggregation linked to the `Item` class. By double clicking on a class in the Diagram or Model Views, you can open its source code in the Visual Studio Editor to view such tags. For example, the source for the `itemList` field in the `Store` class is as follows:

```
///     <link_link>aggregation</link_link>
///     <link_supplierCardinality>0..*<link_supplierCardinality>
///     <Associates>Item</Associates>
```

Creating hyperlinks

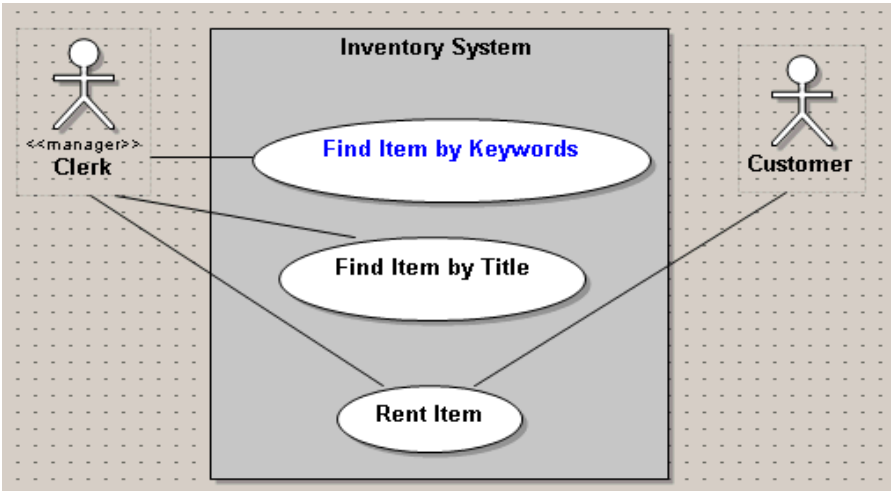
The Video Store has a `findByKeywords` method used for searching for items. The requirements for this method are defined as the use case you created earlier, *Find Item by Keywords*. By using hyperlinks, you can link diagrams and elements to express these types of relationships and record them in the model for others to use.

To create a hyperlink between the use case and the method:

1. Open the Video Store Use Case Diagram.
2. Right click on the **Find Item by Keywords** use case and choose **Hyperlinks > Edit**. The Hyperlinks dialog opens.
3. From the *Available Content* pane, select **Video Store > Store > findByKeywords**.
4. Click **Add** to add the element to the *Selected* pane on the right, and click **OK**.

   The use case with the newly created hyperlink is highlighted in blue font as shown in Figure 14.

Figure 14 Hyperlinked element



5. To test the hyperlink, right click on the use case and choose **Hyperlinks > Video Store.Store.findByKeywords**. The class diagram containing the corresponding method opens with the method selected.

Note Hyperlinks are bi-directional. The hyperlinked method on the class diagram is also highlighted in blue font. You can navigate from the method on the class diagram to the use case diagram and vice versa using the **Hyperlinks** command for the hyperlinked element.

Creating a sequence diagram

Sequence diagrams are used to design the dynamic aspects of an object model.

Together VS .NET provides three modes for creating sequence diagrams:

- A simple “sketch pad”
- Two-way with source code and classes
- Generating a sequence diagram from an existing method

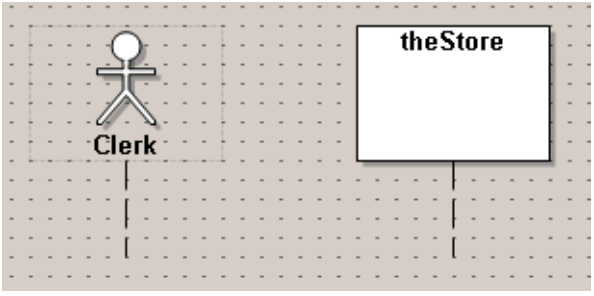
Creating the initial sequence

You can create a sequence diagram in the Video Store project to model some of the steps necessary to rent a video.

To create a sequence diagram:

1. In the Model View, right click on the **Video Store** project node, and choose **Add > Other Diagram**.
2. In the resulting dialog box, choose **Sequence Diagram**.
3. Enter “Rent Movie Sequence” as the name, and click **OK**.
4. From the Toolbox, select **UML Interaction Diagram**. Click the Actor button , and then click the diagram background.
5. Using the in-place editor, enter “Clerk” as the actor name.
6. Using the Toolbox, click the Object button and click to the right of the actor line. Name the object “theStore” as shown in Figure 15.

Figure 15 Clerk actor and the Store object



7. To add a message link, click the Message button and then drag-and-drop from the *Actor lifeline* to *theStore lifeline*.

Associating an object with a class

In sequence or collaboration diagrams you can create associations between objects and classifiers.

Note You can associate an object with all classifiers including, classes, interfaces, structures (in C# projects only), modules (in Visual Basic .NET projects only), enums, or delegates.

Instantiated classifiers for an object can be selected from the model, or the classifiers can be created and added to the model. In the interaction diagrams there are two commands available on the object right-click menu:

- **Add** that creates a new classifier in the model, and
- **Choose class** that provides a list of available classifiers.

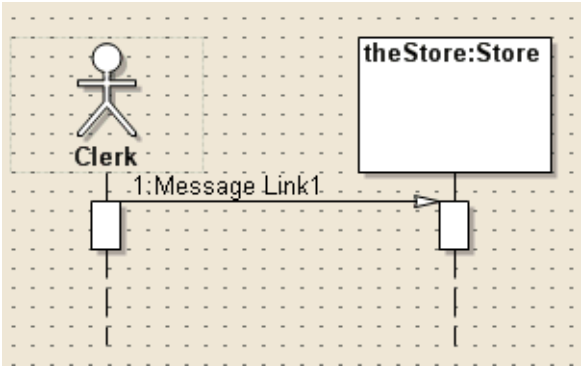
In this example, we will associate `theStore` object with the `Store` class.

To associate an object with a classifier:

1 Right-click on `theStore` object, and select **Choose class** > **Store**. The resulting sequence diagram is shown in Figure 16. Notice that the object name is followed by the name of the associated classifier.

Note After you associate a classifier to an object, you can right click the object and select **Unlink class** from the context menu to remove the association.

Figure 16 theStore object associated with the Store class



Adding a new method to a message link

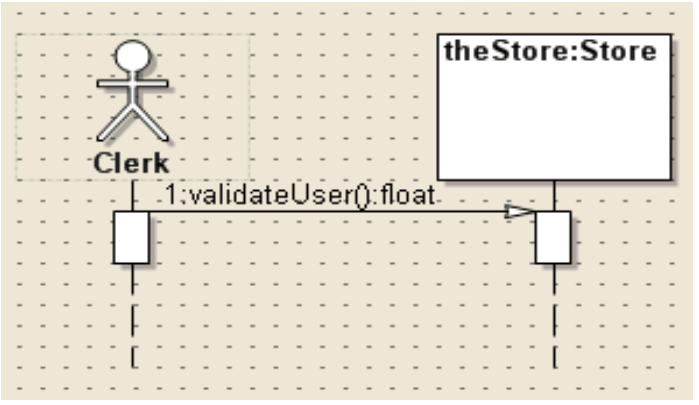
Message links can be associated with the methods of the recipient class. You can select methods from existing methods in the recipient class or create new ones. This is accomplished by two commands provided by the message right-click menu, **Add** and **Choose method**.

In this example, we will create a new method in the `Store` class, `validateUser`.

To add a new method to a message link:

1. Right click on the message link, and choose **Add > Method**. A new method named, `Method1():void` is created in the `Store` class.
2. In the Properties Window for the message link, select the **Operation** field, to rename the new method, and enter, `validateUser():float`.
3. The *Rename Operation* dialog opens indicating that the `validateUser` operation does not exist in the `Store` class. The dialog gives you the option to either create a new method or rename the existing method. Select **Rename** to rename the existing method. The resulting diagram is shown in Figure 17.

Figure 17 validateUser method



You can also use the **Choose method** command on the message link right-click menu to associate the message link with an existing method in the object's recipient class.

Note After you associate a method to a message link, you can right-click the object and select **Unlink method** from the context menu to remove the association.

Generating a sequence diagram from existing source code

You can create sequence and collaboration diagrams and populate them using buttons in the Toolbox. You can also generate sequence diagrams from methods on a class diagram.

Note This feature is available for C# projects only.

In this example, we will generate a simple sequence diagram using the Video Store project.

To generate a sequence diagram:

1. Double click the Video Store *default* class diagram node  . The default class diagram opens in the Diagram View.
2. Right click on the `findByKeywords` method on the `Store` class, and choose **Generate Sequence Diagram**.
3. The *Generate Sequence Diagram* dialog opens. Click **Ok**.

The sequence diagram is generated and opens in the Diagram View.

You can close this sequence diagram without saving any changes to it.

Tip To generate a collaboration diagram from an operation, first generate the sequence diagram and then convert the diagram into a collaboration diagram. For more information, see "Converting between sequence and collaboration diagrams".

Converting between sequence and collaboration diagrams

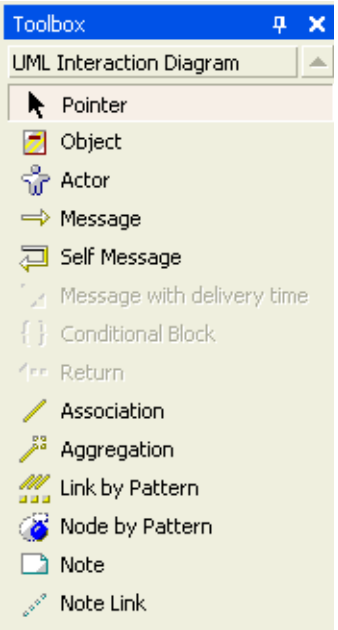
You can convert between sequence and collaboration diagrams. However, when you create a new diagram in Together VS .NET using the Add New Diagram dialog, you must specify whether it is a sequence or collaboration diagram.

To convert the Rent Movie sequence diagram to a collaboration diagram:

1. Open the Rent Movie sequence diagram.
2. Right click on the diagram background, and choose **Show as Collaboration** from the context menu. The collaboration diagram opens.
3. Open the Toolbox. From the View menu, choose **Toolbox**.
4. In the Toolbox, click **UML Interaction Diagram** to display the collaboration diagram elements as shown in Figure 18.

Notice that the sequence and collaboration diagrams share the UML Interaction Diagram toolbox, however, only certain diagram elements are enabled for each type of diagram.

Figure 18 Collaboration diagram elements



5 Right click on the diagram background, and choose **Layout > Do Full Layout**.

6 To convert back to the sequence diagram, right click on the collaboration diagram background, and choose **Show as Sequence**. The sequence diagram view displays in the diagram.

Using the Model View

Use the Model View as your primary navigational view in Together VS .NET to manage diagram elements. Using the context menus in the Model View, you can open, create, and delete diagrams and diagram elements.

The Model View consists of a number of icons and context menu commands that you should become familiar with. Explore the context menus of the different elements and diagrams as you encounter them.

Using the context menu of your root project node, you have access to the following features of Together VS .NET:

- Adding diagrams to your project
- Adding diagram elements such as, namespaces, classes, and interfaces
- Refactoring code
- Running audits against C# projects

Viewing references

The Model View enables you to view class diagrams for references included in your project. You can add references to your project using the Solution Explorer.

To view the MsCorLib.dll in the Video Store project:

1. Expand the *References* node and the *MsCorLib.dll* node in the Model View.
2. Right click on the default diagram, and choose **Open Diagram**. The default diagram opens in the Diagram View.

You can expand the Microsoft and System folders to view other class diagrams as well.

Use the context menu in the Model View or the File main menu (**File > Print** or **File > Export Diagram to Image**) to print diagrams or export diagrams to image files.

Showing expandable diagram nodes

By default, the diagram nodes in the Model View are expandable. You can control whether you can expand nodes in the Model View to show their contents.

To make diagram nodes expandable:

1. From the Tools menu, choose **Options**. The Options dialog opens.
2. Click the *Together VS .NET* folder to view Together VS .NET-specific configuration options. The General options display.
3. Select *Model View* from the list.
4. Click on the *Show diagram nodes expandable* field. A drop down arrow displays.
5. Click the drop down arrow, and select *True* from the list.
6. Click **OK** to close the dialog and apply the changes.

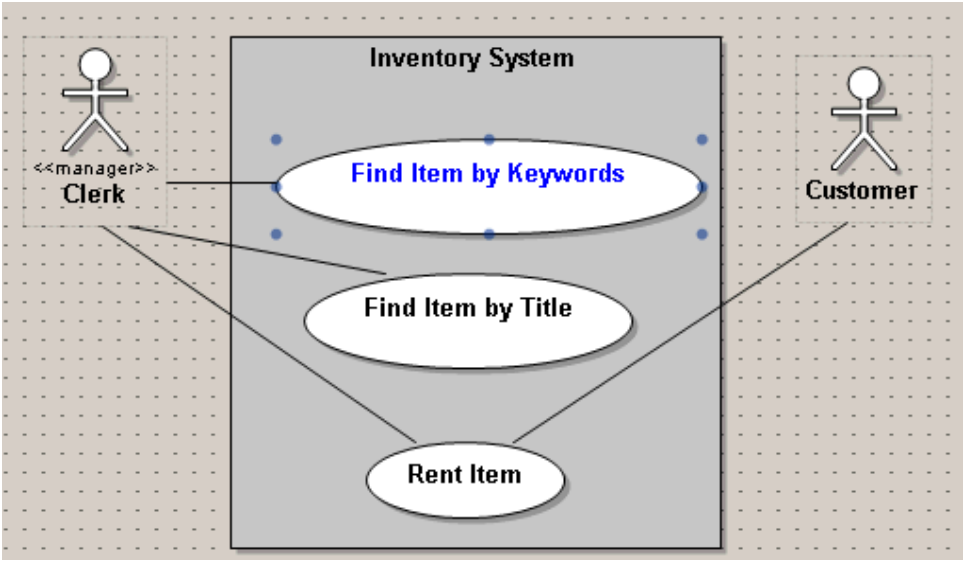
The Model View diagrams display as expandable nodes. Expand the node for the *Video Store Use Case* diagram to view its elements.

Finding diagram elements

When selecting a diagram element in the Model View, the element is not automatically selected in the Diagram View. To show an element selected from the Model View in the Diagram View, right click on the element in the Model View, and choose **Select on Diagram**. Using this command, you can open the corresponding diagram of the element, and highlight the element on it as shown in Figure 19.

Similarly, while working in the Diagram View, you can use the **Synchronize Model View** command to navigate directly to an element in the Model View. Right click on a diagram element, and choose **Synchronize Model View** from the context menu.

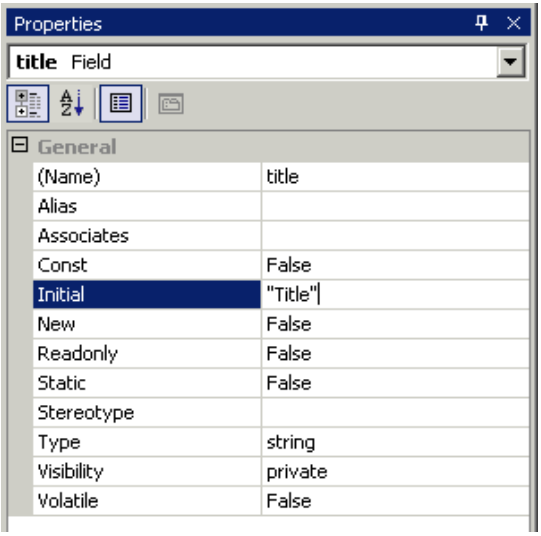
Figure 19 Highlighted diagram element



Using the Properties Window

You can use the Properties Window to view and edit values such as alias, name, visibility, and others as shown in Figure 20.

Figure 20 Properties Window



To add details to the Item class:

1. Open the Video Store class diagram, and select the “title” field of the Item class.
2. Open the Properties Window by choosing **Properties Window** from the View menu. Alternatively, use **F4**.
3. In the *Initial* field, enter “Title” (include the quotes), as shown in Figure 20.

The source code automatically updates to reflect this change. You can easily navigate from the diagram view to the source code.

To open the source code editor for the title field:

1. Right click the `title` field of the `Item` class on the diagram.
2. Choose **Go to Definition** from the context menu.

The source code editor opens, highlighting the appropriate code. Notice that the initial value for the title field is defined as specified within the Properties Window.

Using the Overview

The overview feature of the Diagram View provides a thumbnail view of the current diagram. The **Overview** button is located in the bottom right corner of every diagram, as shown in Figure 21.

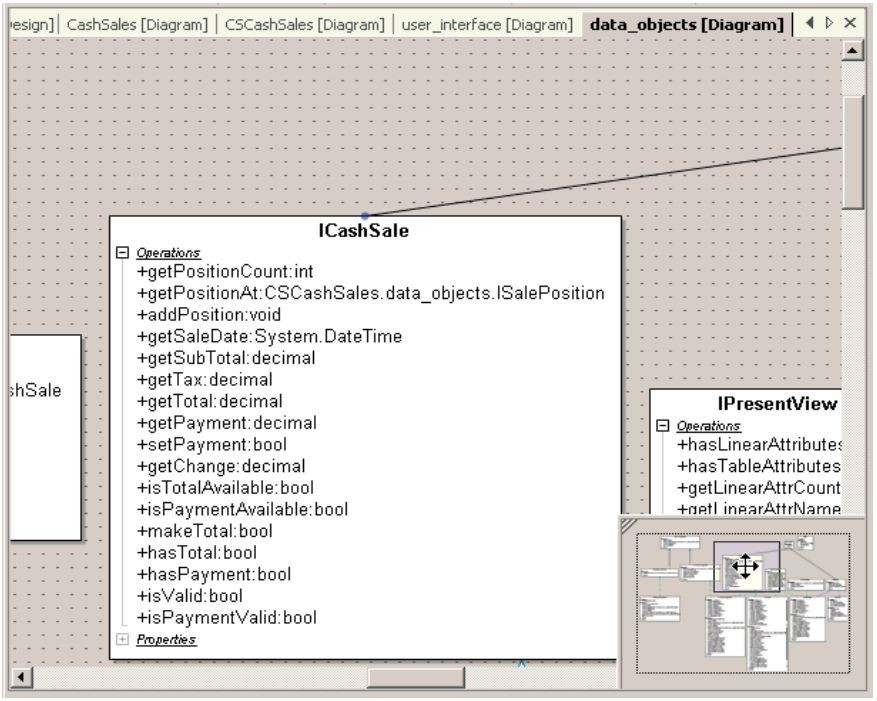
Figure 21 Overview button

To use the overview feature:

1. On the Video Store class diagram, click the **Overview** button. The pane expands to show a thumbnail image of the current diagram.
2. Use the mouse to click on the shaded area and drag it. This is a convenient way to scroll around the diagram.
3. Alter the size of the Overview pane by clicking on the upper left corner of the pane and dragging to resize it.

   The Overview pane automatically closes when you select an element on the diagram. Figure 22 shows the overview pane for a large class diagram.

Figure 22 Thumbnail of a large class diagram



Managing diagram views

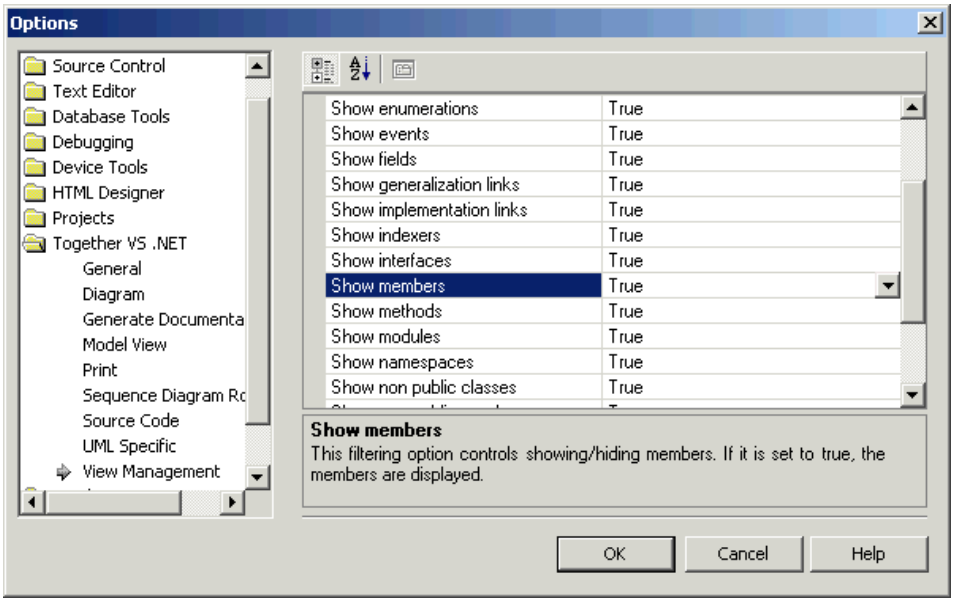
View management controls which diagrams and elements are displayed in Together VS .NET.

Hiding/Showing Information

When dealing with large projects, the amount of information shown on a diagram can become overwhelming. In Together VS .NET, you can selectively show or hide information.

For global control over the diagram view, you can use the filters in the Options dialog. From the Tools menu, select **Options**. Click the **Together VS .NET** folder, and select the **View Management** node as shown in Figure 23.

Figure 23 Options dialog for view management



The filters shown in Figure 23 are global filters. To specifically filter members, you can set the *Show members* property to *False*.

To filter members:

1 Click on the *Show members* field.

2 Click the drop down arrow, and select *False.*

3 Click **OK**.

This results in disabling the Fields, Methods, Non-public Members filters as well as any inner classifiers.

Since inner classifiers are treated as members of the container element, the following filters do not filter inner classifiers:

- Show classes
- Show delegates
- Show enumerations
- Show interfaces
- Show structures

Note Structure elements are currently available in C# projects only.

Using the Show Hidden dialog

You can also use the Show Hidden dialog to selectively show or hide information on your diagrams. The next example demonstrates how to hide and show information on the default class diagram of the Video Store project.

To hide/show elements:

1. Double click the Video Store *default* class diagram node  in the Model View. The default class diagram opens in the Diagram View.
2. In the Diagram View, right click on the `Store` class, and choose **Hide** from the context menu. The `Store` class disappears from the diagram.
3. To show the `Store` class, right click on the diagram background, and choose **Show/Hide** from the context menu. The *Show Hidden* dialog opens.
4. The `Store` class displays in the *Hidden Elements* list on the right of the dialog. Select the `Store` class, and click **<<Remove**, and then click **Ok**. The `Store` class displays on the diagram again.

CHAPTER

3

Using Together VS .NET Features

The information in this chapter is specific to using the special features of Borland Together Edition for Microsoft Visual Studio .NET (Together VS .NET). For more comprehensive information, refer to the online help for Together VS .NET.

This chapter includes the following topics:

Using and creating patterns

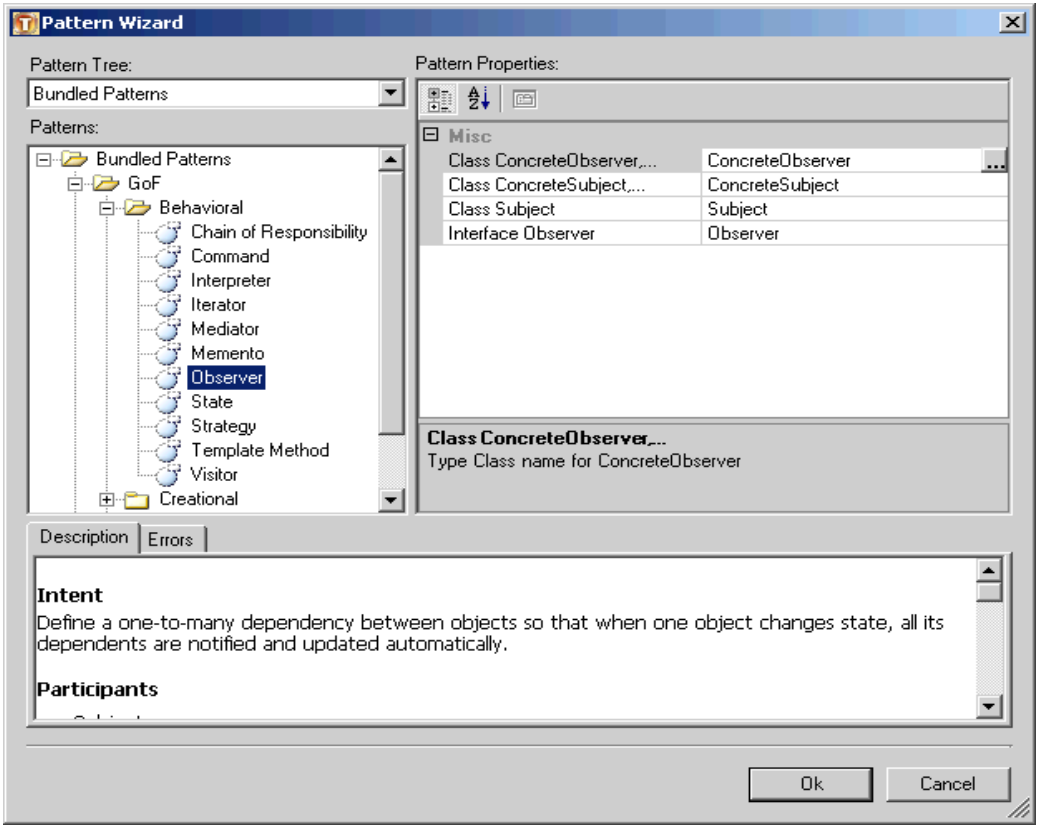
Together VS .NET provides support of frequently used patterns such as the GoF patterns. You can use patterns to create or modify existing links and classes.

For the following example, assume that the `Item` class requires a dynamic system so that after an Item has been returned, the counter is updated and other systems are notified. The Observer pattern is useful for designing such a system. To apply the Observer pattern information to the `Item` class, we will use the Pattern Wizard.

To use the Pattern Wizard:

1 Right click on the Video Store class diagram background, and choose **Create by Pattern**. The Pattern Wizard opens as shown in Figure 24.

Figure 24 Pattern Wizard

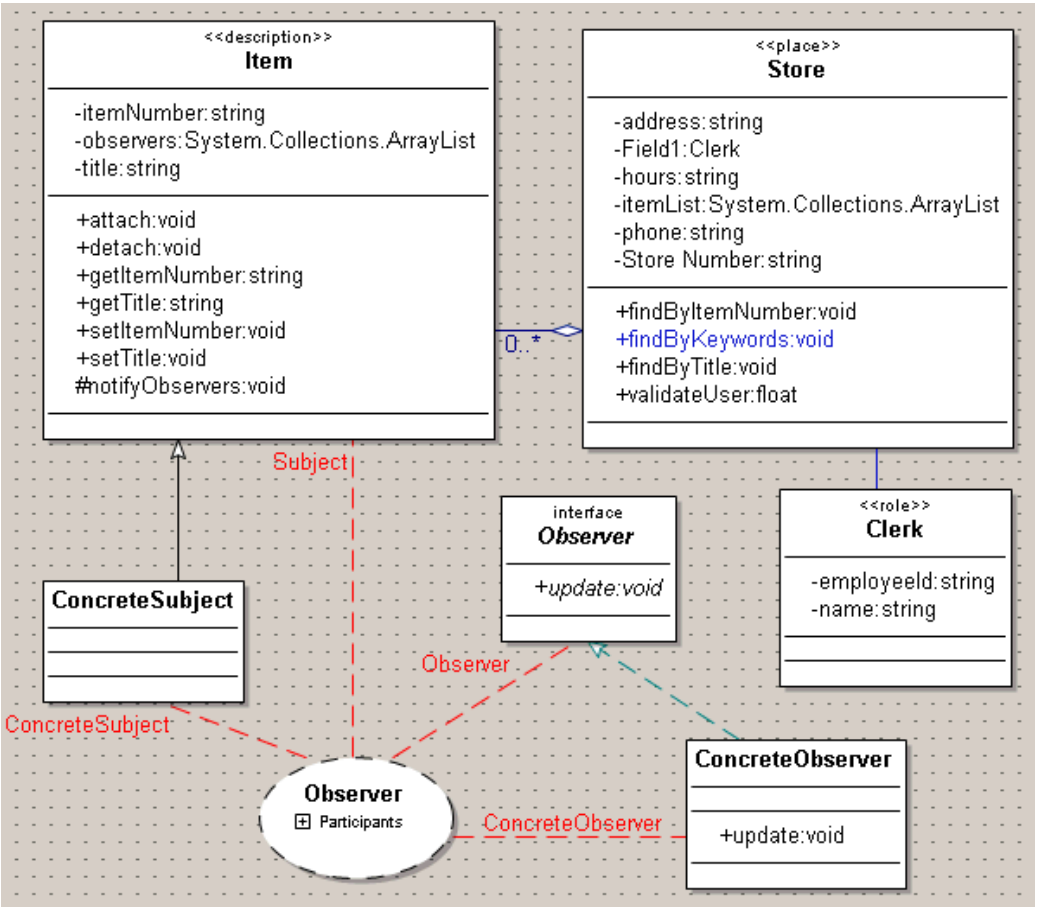


2 From the *Patterns pane* on the left, choose the **GoF > Behavioral > Observer** pattern.

3 In the *Pattern Properties* pane on the right, change the *Class Subject* name to `Item`.

4 Accept the other default properties settings, and click **OK**. The diagram is updated as shown in Figure 25.

Figure 25 Applying the Observer pattern



`Item` is updated with the notification and observer methods (`attach` and `detach`). The other classes and interfaces have been created ready for use. Together VS .NET recognizes the pattern and visualizes the element on the diagram as an oval shape. In addition, it lists the participants of the pattern and the pattern links. You can expand the *Participants* node on the oval Observer element to view pattern participant information.

The pattern elements on the diagram have specific pattern actions that you can choose relevant to the pattern. Right click on the oval Observer element, and choose **Add** to view the specific pattern actions available.

Running quality assurance

You can run quality assurance (QA audits) on your **C# projects** to check the quality of your code against a set of predefined measurements.

Note The QA audits feature is available for C# projects only.

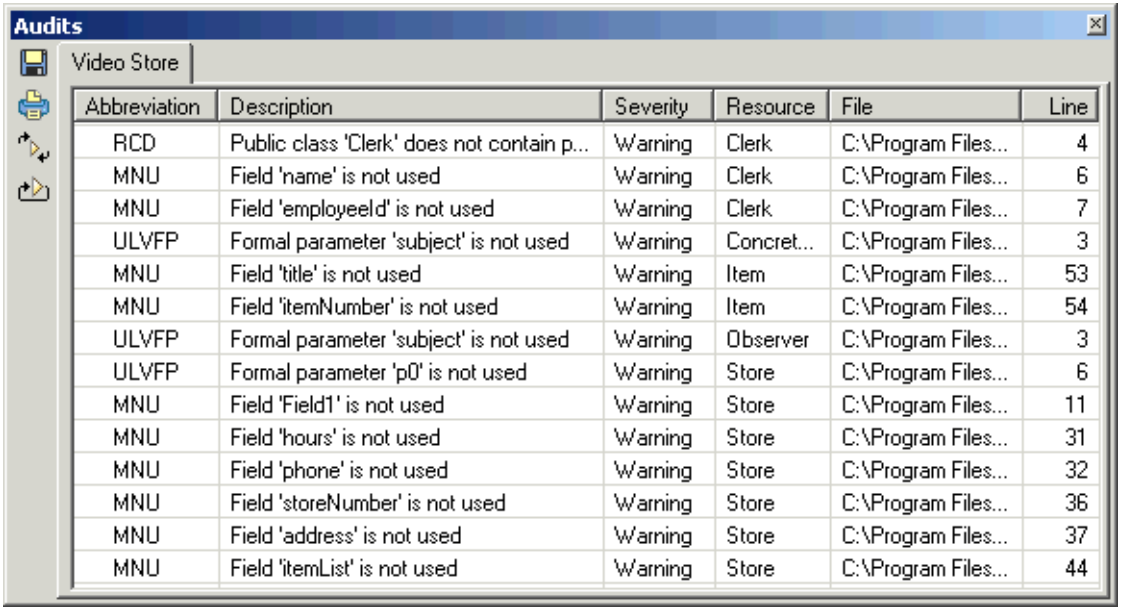
Running audits

Audits help you unobtrusively enforce company standards and conventions and improve what you do. When you run audits, you select specific rules to which your source code should conform to. The results display the violations of those rules so that you can examine each problem and decide whether to correct the source code or not. Together VS .NET provides a wide variety of audits to choose from, ranging from design issues to naming conventions, along with online descriptions of what each audit looks for and suggestions on how to fix violations.

To run audits on the Video Store project:

1. Right click on the Video Store class diagram background, and choose **QA Audits**. The Audits dialog opens.In the *Scope* drop down list, choose Model.
2. On the left side of the dialog, expand the nodes of the audit categories (Coding Style, Declaration Style, and so on) to view the available audits, and check (or clear) the appropriate checkboxes to indicate which audits to run. As you click on an audit, its description displays in the lower pane of the dialog.
3. For each audit, the severity level and other audit-specific options are displayed in the right-hand panel of the Audits dialog box. Change the settings if necessary.
4. For this example, accept the default settings, and click **Start**. The *Operation in progress* dialog opens displaying a status bar that indicates the progress completed, until the process finishes. The Audit results are shown in Figure 26.

Figure 26 Audits view (your results may differ)



Abbreviation	Description	Severity	Resource	File	Line
RCD	Public class 'Clerk' does not contain p...	Warning	Clerk	C:\Program Files...	4
MNU	Field 'name' is not used	Warning	Clerk	C:\Program Files...	6
MNU	Field 'employeeId' is not used	Warning	Clerk	C:\Program Files...	7
ULVFP	Formal parameter 'subject' is not used	Warning	Concret...	C:\Program Files...	3
MNU	Field 'title' is not used	Warning	Item	C:\Program Files...	53
MNU	Field 'itemNumber' is not used	Warning	Item	C:\Program Files...	54
ULVFP	Formal parameter 'subject' is not used	Warning	Observer	C:\Program Files...	3
ULVFP	Formal parameter 'p0' is not used	Warning	Store	C:\Program Files...	6
MNU	Field 'Field1' is not used	Warning	Store	C:\Program Files...	11
MNU	Field 'hours' is not used	Warning	Store	C:\Program Files...	31
MNU	Field 'phone' is not used	Warning	Store	C:\Program Files...	32
MNU	Field 'storeNumber' is not used	Warning	Store	C:\Program Files...	36
MNU	Field 'address' is not used	Warning	Store	C:\Program Files...	37
MNU	Field 'itemList' is not used	Warning	Store	C:\Program Files...	44

Working with the audit results view

Using the audit results view, you can perform several tasks, such as:

- Opening the corresponding source code for an audit violation.
- Viewing the description of an audit violation.
- Printing and saving audit results.
- Sorting or grouping audit results.
- Refreshing (recalculating) the audit violations that are currently displayed.
- Restarting the audit calculation.

Notice that the audit results view is a dockable window. The audit results open initially as a free-floating window; however, it can be free-floating or docked to a docking area. The docking areas for the audit results are comprised of any of the four borders of the Visual Studio .NET window -- position the audit results window according to your preferences.

Opening source code and viewing an audit description

The results report is tightly connected with the diagram elements and the source code. Using the report, you can navigate to the specific location in the source code where the violation actually takes place.

To open the source code for an audit violation and view an audit description:

1. In the Audit results view, double click the second entry for *Field 'name' is not used*. Together VS .NET opens the source in the Visual Studio Editor, and highlights the line of code.
2. In the Audit results view, right click *Field 'name' is not used*, and choose **Show Description.** This opens a description of the *Member is Not Used (MNU)* audit.

Printing audit results

You can print the entire table of audit violations or select specific rows and columns to print.

To print audit results:

1. Select the rows of the table that you want to print. If you want to print the entire list, you do not need to select anything.
2. Click the Print button  on the toolbar. The *Print Audit dialog* opens.
3. Choose the scope of the results to print using the *Select View* combobox (*All Results*, *Active Group*, or *Selected Rows*). Audit results display in tabbed-pages in the audit results report view. You can *group* and *ungroup* the results using the **Group by** command on the report view context menu. **Note:** Unless the results have been grouped using the **Group by** command, the Active Group option is not enabled in the dialog. The possible view options are:
   - *All Results* - If the results are *grouped*, choosing *All Results* prints a report for all groups in the current tabbed-page. If the results are not *grouped*, then all results print for the current tabbed-page.
   - *Active Group* - If the results are *grouped*, then you can select a group in the current-tabbed page, and the printed report contains the results from the selected group.
   - *Selected Rows* - You can select single or multiple rows in the audit results report view. Choosing *Selected Rows* prints a report for such selections.
4. If desired, specify the print zoom factor in the *Print zoom* field, or check *Fit to page* if you want to print the results on a single page. If checked, the *Print zoom* field is disabled.
5. If necessary, adjust the page and printer settings:
   - Click the *Print* combobox, and choose the *Print dialog* command to select the target printer.
   - Choose, **Tools | Options | Together VS.NET - *Print*** from the main menu, and use the *Options dialog* to set up the paper size, orientation, and margins.
     - Click the drop-down arrow to the right of the Preview option to open the preview pane. Use the *Preview zoom (auto)* slider, or *Auto*

preview zoom checkbox as required. Click the upward arrow to the right of the Preview option to close the preview pane.

6 Click **Print** to open the system print dialog, and send the file to the printer.

Saving audit results

Export audit results to an XML or HTML file so you can share them with team members or review them later.

To save audit results:

1 Select the rows of the table that you want to save. If you want to save the entire list, you do not need to select anything.

2 Click the Save button  on the toolbar.

3 In the *Save Audit Results* dialog that opens, choose the scope of the results to export using the Select View combobox (*All Results*, *Active Group*, or *Selected Rows*). Audit results display in tabbed-pages in the audit results report view. You can *group* and *ungroup* the results using the **Group by** command on the report view context menu. **Note:** Unless the results have been grouped using the **Group by** command, the Active Group option is not enabled in the dialog. The possible view options are:

- *All Results* - If the results are *grouped*, choosing *All Results* generates a report for all groups in the current tabbed-page. If the results are not *grouped*, then all results for the current tabbed-page are exported.
- *Active Group* - If the results are *grouped*, then you can select a group in the current-tabbed page, and the exported report contains the results from the selected group.
- *Selected Rows* - You can select single or multiple rows in the audit results report view. Choosing *Selected Rows* generates a report for such selections.

4 In the *Select Format* combobox, select the format for the exported file.

- *XML*- Generates an XML-based report.
- *HTML* - Generates an HTML-based report.
  Selecting HTML format activates the checkboxes, *Add Description* and *Launch Browser*.
  - *Add Description* - This saves the audit descriptions in a separate folder with hyperlinks to the descriptions from the results file.
  - *Launch Browser* - This option opens the generated HTML file in the default viewer.

5 Click **Save** to save the results in the specified location.

Sorting or grouping audit results

When viewing audit results, you might want to compare and organize the items in the results report. You can sort and group the results as follows:

- ***Sorting by one column:*** To sort all the items according to the values for a specific column, click on the column heading. Click once to sort in ascending order. Click a second time to sort in descending order.
- ***Grouping audit results:*** To group items according to the current column, right-click in the Audit results table and choose **Group By**. This enables you to organize the results by changing the relationship of rows and columns. To ungroup the results, right click on the table, and choose **Ungroup**.

Refreshing and restarting audits

You can update or refresh the results table using the Toolbar.

- Click Refresh  to recalculate the results that are currently displayed.
- Click Restart  to open the Audits dialog, where you can change the settings as necessary and run the audits again. The new results replace the results that are currently displayed.

Close the Audits view by clicking the Close button  in the upper-right corner of the dialog.

Choosing specific elements to run audits against

Using the Audits feature of Together VS .NET, you can selectively determine which classes to run a set of audits against.

To run audits on a particular class element:

1. Go to the Video Store diagram, right click on the `Item` class and choose **QA Audits**. This restricts the *Scope* of the audit to the `Item` class.
2. Accept all other default settings, and click **Start**. The Audit results view opens. As you run audits on your project, the results display in the Audit results view in a tabbed-page format. The most recent audits ran have focus.
3. You can close the audit results (individual tabbed-pages) by right clicking on the corresponding tab, and choosing **Close**. Click the Close button  in the upper right corner of the results dialog to close the audit results.

Refactoring

Refactoring means rewriting existing source code with the intent of improving its design rather than changing its external behavior. The focus of refactoring is on the structure of the source code, changing the design to make the code easier to understand, maintain, and modify.

Together VS .NET supports the following refactoring commands:

- **Rename:** Rename a code-generating element. Available in the Diagram and Model Views and the Visual Studio Editor.
- **Move:** Move a static method to a different class. Available in the Diagram and Model Views and the Visual Studio Editor.
- **Extract Interface:** Create a new interface from an existing classes, structure, method, property, event, or indexer. Available in the Diagram and Model Views and the Visual Studio Editor.
- **Extract Superclass:** Create a superclass from an existing class, interface, method, property, event, field, or indexer. Available in the Diagram and Model Views and the Visual Studio Editor.
- **Pull Members Up:** Move selected members higher in the class hierarchy. Available in the Diagram and Model Views and the Visual Studio Editor.
- **Push Members Down:** Move selected members lower in the class hierarchy. Available in the Diagram and Model Views and the Visual Studio Editor.
- **Safe Delete:** The Safe Delete command allows you to view whether there are any usages of an element before deleting it. Available in the Diagram and Model Views and the Visual Studio Editor.
- **Change Parameters:** Change existing parameters or create new parameters in a selected method. Available in the Diagram and Model Views and the Visual Studio Editor.
- **Extract Method:** Extract a set of complete statements in the Visual Studio Editor to create a new method. Available in the Visual Studio Editor only.
- **Inline Variable:** Create an inline variable by selecting a local variable in the Visual Studio Editor. Available in the Visual Studio Editor only.
- **Introduce Field:** Create a new field from a single, complete expression. Available in the Visual Studio Editor only.
- **Introduce Variable:** Create a new variable from a single, complete expression. Available in the Visual Studio Editor only.

The commands are available from the **Refactoring** main menu or via the **Refactoring | <menu command>** context menu for class diagram code-generating elements. For more complete details on these commands, refer to Together VS .NET online help.

Renaming

You can rename any code element: class, interface, method, field, parameter, local variable, and so on. Together VS .NET propagates the name changes to the dependent code in your project files.
To use the rename command on the Item class:

1. Right click on the `Clerk` class in the Diagram View, and choose **Refactoring > Rename class "Clerk"**. The Rename dialog opens.
2. For the new name, enter `Employee` in the *New name* field of the dialog.
3. Be sure that the option, *View references before refactoring*, is checked. With this option activated, upon clicking **Rename**, the Refactoring window opens allowing you to review the refactoring before committing to it. If the option, *View references before refactoring*, is cleared, then upon clicking **Rename**, the Refactoring window opens, and the renaming is completed.
4. Click **Rename**. The Refactoring window opens. The Refactoring window displays the results of using the rename command. The Rename command will change the name of the `Clerk` class and any usages found for the `Clerk` class. For this example, only one usage displays because a relationship exists between the `Clerk` and `Store` classes. The relationship is represented in the source code of the `Store` class by the field `Clerk`.
5. Click the perform refactoring button  on the left of the dialog to complete the renaming.
6. Close the Refactoring window. Click the Close button in the upper right corner of the window.

Working with the Refactoring window

The Refactoring window is a dockable window. It opens initially as a free-floating window; however, it can be free-floating or docked to a docking area. The docking areas for the window are comprised of any of the four borders of the Visual Studio .NET window -- position the Refactoring window according to your preferences.

Each time that you use a Refactoring command for the same project, the results display in a tabbed-page format with the most recent results having focus in the window. You can close the results by right clicking on the corresponding tab, and choosing:

- **Close:** Closes the current tab in focus.
- **Close All:** Closes all tabs and the Refactoring window.
- **Close All But This:** Closes all tabs except for the tab currently in focus.

To close the Refactoring window (leaving all tabs open), click the Close button in the upper right corner of the window.

Generating documentation

Together VS .NET features a UML documentation wizard that you can use to generate HTML documentation for the Video Store project.

To generate HTML documentation for a project:

1. Select **Tools | Generate Documentation** from the main menu. The Generate Documentation dialog opens, as shown in Figure 27.
2. Select your preferred *Scope* and *Options* settings, and click **OK** to generate documentation.
3. You are prompted to create the `out/doc` directory in the Video Store project. Click **Yes**.

By default, the Generate Documentation wizard creates documentation for your entire project. You can limit the scope of the documentation to a smaller set by choosing a different *Scope* option.

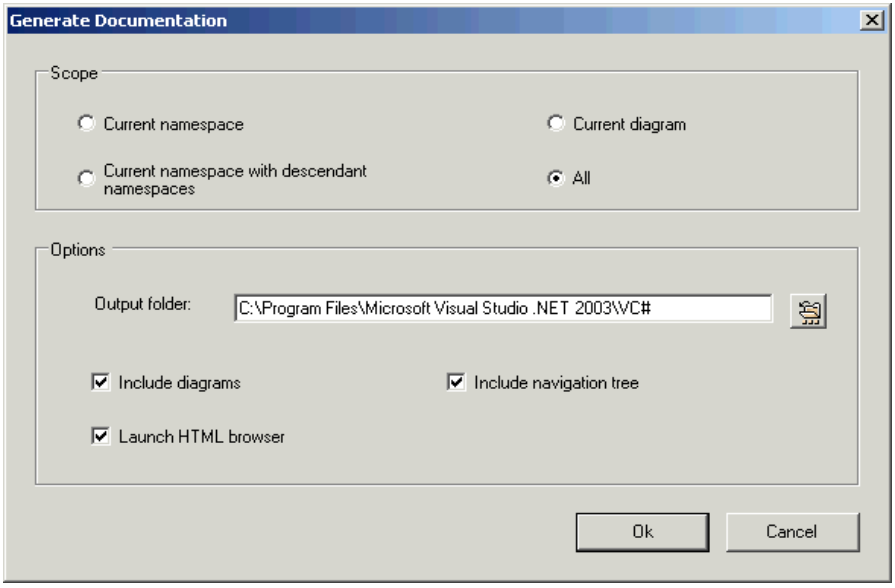
The *Scope* section at the top of the dialog has radio buttons to indicate what parts of the project should be parsed and included in the generated documentation.

- **Current namespace:** Generated output includes only the current namespace selected in the Model View.
- **Current namespace with descendant namespaces:** Generated output includes the current namespace selected in the Model View and any descendant namespaces under it.
- **Current diagram:** Generated output for the current diagram that is in focus in the Diagram View.
- **All:** Generated output covers the entire project.

The *Options* section of the dialog has options to specify the destination and other optional actions.

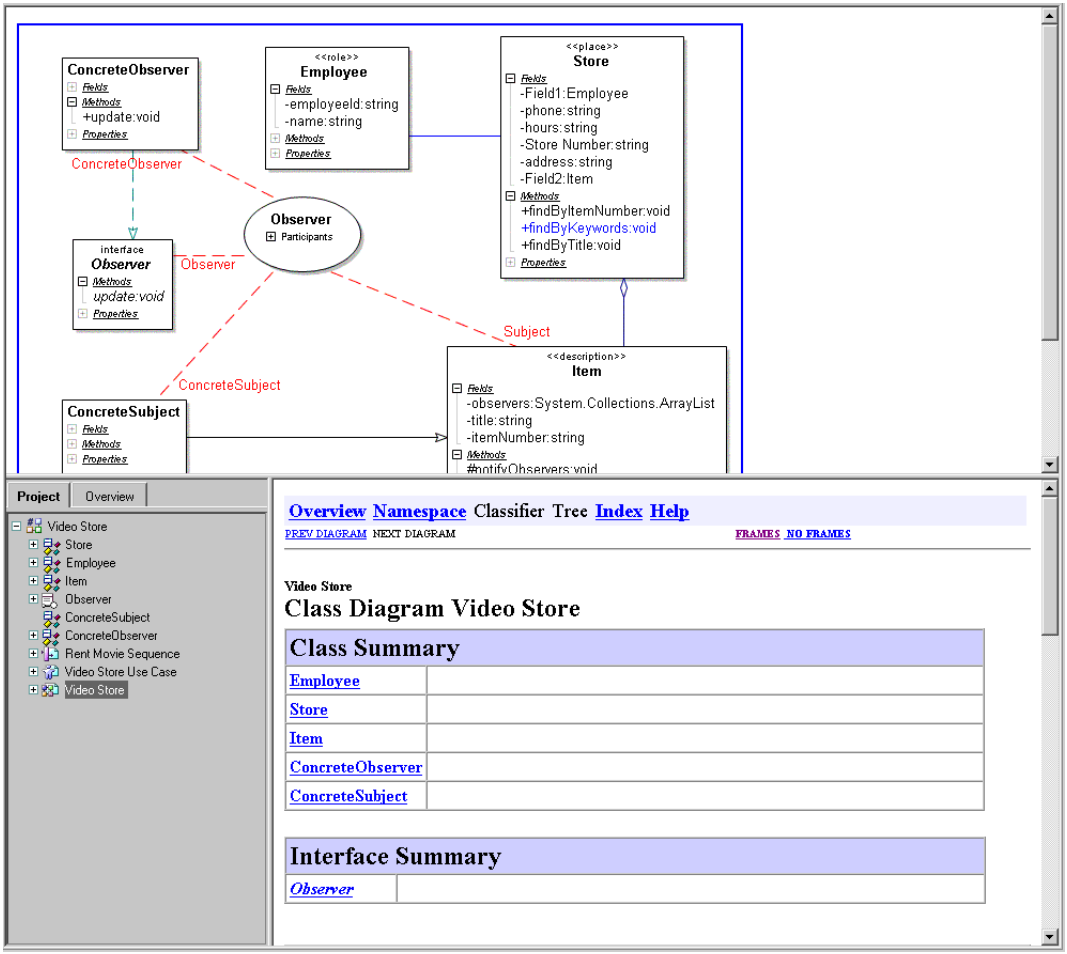
- **Output folder:** Enter the location for the generated output, or select from the file chooser.
- Checkboxes:
  - **Include diagrams:** Check to include diagram images in the output.
  - **Include navigation tree:** Check to include a navigation tree in the output.
  - **Launch HTML browser:** Check to load the documentation in the default web browser for your operating system. This starts the application if necessary. If you do not select this option, you can open the documentation later by navigating to your designated output folder and accessing the `index.html` file.

Figure 27 Generate Documentation dialog



When Together VS .NET finishes generating the documentation, it opens in the default web browser for your system. The browser opens with a frameset to display the generated documentation. Expand the `Video Store` node in the tree in the lower left frame. Your browser displays a page similar to Figure 28. Notice that clicking a class name in the lower left frame opens the documentation in the lower right pane.

Figure 28 Resulting documentation



You can further explore the generated documentation with the following exercises:

- Use the **Project** tab to navigate through the project.
- Click the `Store` class to display its corresponding documentation.
- Click `findByKeywords` to jump to that section in the documentation.
- Use the hypertext to navigate through the documentation and access the index.

Exporting and Importing XMI projects

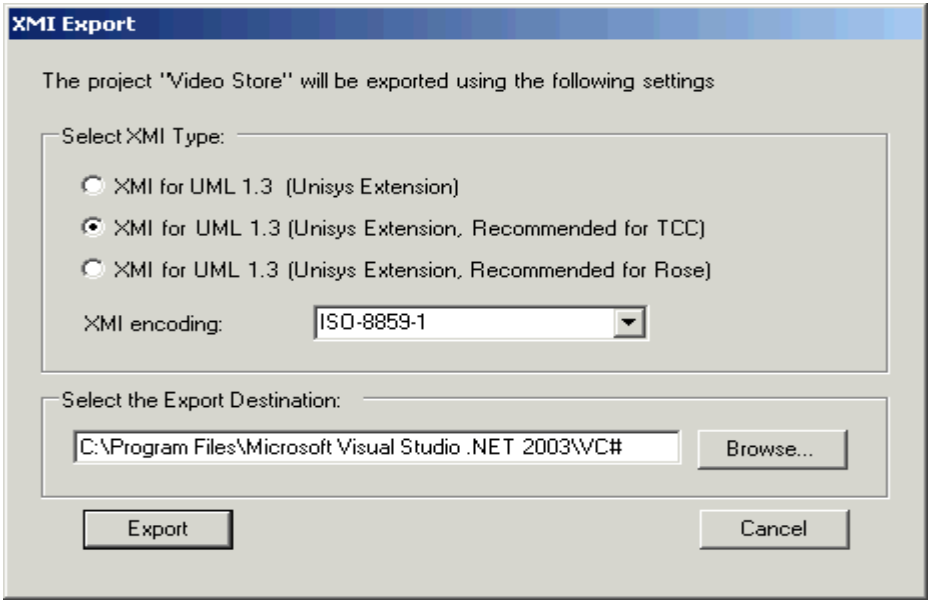
Together VS .NET also features XMI import and export capabilities. You can import a model described by an XMI file into a Together VS .NET project generating source code. You can also export your Together VS .NET project to an XMI file making the model information available outside of Together VS .NET.

Exporting XMI projects

To export a project as an XMI file:

1 In the Model View, right click on the Video Store project and choose **Export Project to XMI**, or choose **File | Export Project to XMI** from the main menu (you must have a diagram open or have the project root node selected in the Model View). The XMI Export dialog opens as shown in Figure 29.

Figure 29 XMI Export dialog



2 In the resulting dialog box, you can select the XMI Type information and export destination for the XMI file. For this example, accept the defaults. Click **Export**.

3 A confirmation dialog asks if you want to create the new directory. Click **Yes.**

4 In the directory structure of your system, locate the new XML file: `<project_directory>\out\xmi\Video Store.xml`.

Importing XMI projects

Though this guide does not import an existing XMI file, the process is similar to exporting.

To import an XMI file into a project:

1. In the Model View, right click on the Video Store project and choose **Import Project from XMI**, or choose **File | Import Project from XMI** from the main menu (you must have a diagram open or have the project root node selected in the Model View). The XMI Import dialog opens as shown in Figure 30.
2. Browse for the source file.
3. Click **Import**.

Figure 30 XMI Import dialog