

codeobjects



CORADS™

CodeObjects Rapid Application Development Studio

Tutorial 1

Fundamentals

CodeObjects, Inc.

Tutorial 1: Fundamentals

© 2011 CodeObjects, Inc. All rights reserved.

No part of this book may be reproduced, stored in a retrieval system, or transmitted by any means electronic or otherwise without written permission from the publisher.

We have made every effort to make this book as complete and accurate as possible, but no warranty of fitness is implied. The information is provided “as is,” and the publisher assumes no liability for any losses or damages arising from the use of this information. The content included in CORADS Tutorials is intended solely for training purposes, and may or may not depict the specific features in the CORADS package as installed on your system. CodeObjects reserves the right to modify CORADS features at any time.

CORADS is the only software certified for customizing applications provided by CodeObjects. You are allowed to use CORADS only to customize the applications provided by CodeObjects. The CORADS feature set received in your software package is subject to your CORADS license agreement.

CORADS is a trademark of CodeObjects, Inc. WebSphere is a registered trademark of IBM. Other trademarks are the property of their respective owners.

This manual is confidential and proprietary information provided by CodeObjects, Inc. for clients only.

CodeObjects Inc.
1381 McCarthy Blvd.
Milpitas, CA 95035 USA
<http://www.codeobjects.com>

Table of Contents

Introduction	1
Description of the Tutorial Application	1
Before You Begin: Environment Setup.....	1
Servers	2
Prerequisites	3
What You Will Create	3
User Interface Terminology	4
Buttons.....	4
Screen Layout.....	4
Hierarchical Trees and Command Sequences	6
Saving Your Work	6
Setting up the Application.....	7
Enable Automatic Builds.....	7
Set the Modules Directory	8
Open CodeObjects Perspective	9
The Metadata Pane	10
The Editor Pane.....	12
The Property Pane	14
Create a Folder for Your Application	14
Create Modules for the Tutorial Application	15
Connect to the Database (Create an Eclipse Connection Profile)	17
Create a New Driver Definition	19
Test the Connection	22
Defining the Data Layer	24
High Level Metadata in the Data Layer	24

Displaying the Metadata	25
Create a Data Table by Importing from the Database	25
Double Checking your Results	27
Create an Entity for the Data Table.....	28
Define a Data Object for your Entity	32
Create a Data Object	32
Defining the Services Layer	35
High Level Metadata in the Services Layer	35
Setting up for Server Deployment.....	36
Metadata for Application and System	36
Set up the Deployment Folder.....	37
Switch to the Package Explorer	37
Create Deployment Folders.....	38
Create the Logging Configuration File	40
Set up Security.....	43
Create the User Credentials File.....	43
Create User Roles.....	43
Create the System Metadata	47
Add Modules to the System	49
Add Services to the System Model	50
Set up Security for the System Model.....	52
Create the Application Metadata.....	55
Set Properties of the Application	56
Deploying the Server	59
Create a Deployable Web Archive File	59
Test the Deployment Files	61
Copy the File	61
Verify the AllDefect Data Object's WSDL	61
Verify the Authentication Service's WSDL.....	61
Defining the User Interface Layer.....	62
High Level Metadata in the Client/UI Layer	62
Create a View	62
Create a Widget.....	64

Deploying the Whole Application	66
High Level Metadata in the Deployment Layer for Clients	66
Create a Client (Metadata)	66
A Note on Errors in CORADS/Eclipse.....	70
Prepare the Deployment for the Client Module.....	71
Deploy and Run the Whole Application	72
Create a Defect	74
Search for a Defect.....	75
Matching Metadata with Results	76
Checking the Application into Source Control	87
Before Checking In.....	87
Files to Check In	88
Check in Your Files	88
Previewing the Complete Application.....	89
Running the Finished Tutorial Files	89
Summary	91
Understanding the Module Structure	92
Understanding the Metadata and How to Work with It	93

Introduction

These tutorials guide you through the use of CORADS to build an entire CodeObjects application from start to finish.

Description of the Tutorial Application

In these tutorials, you will build a simple defect tracking application for recording bug reports in software development projects. The defect system has two kinds of users, *developers* and *testers*, who perform the following operations:

- **Testers** log defects/bugs, describe the defects, and assign the defects to specific developers.
- **Developers and testers** update the system when the problems are addressed.

In this first tutorial, you will build the basic defect system with one data table:

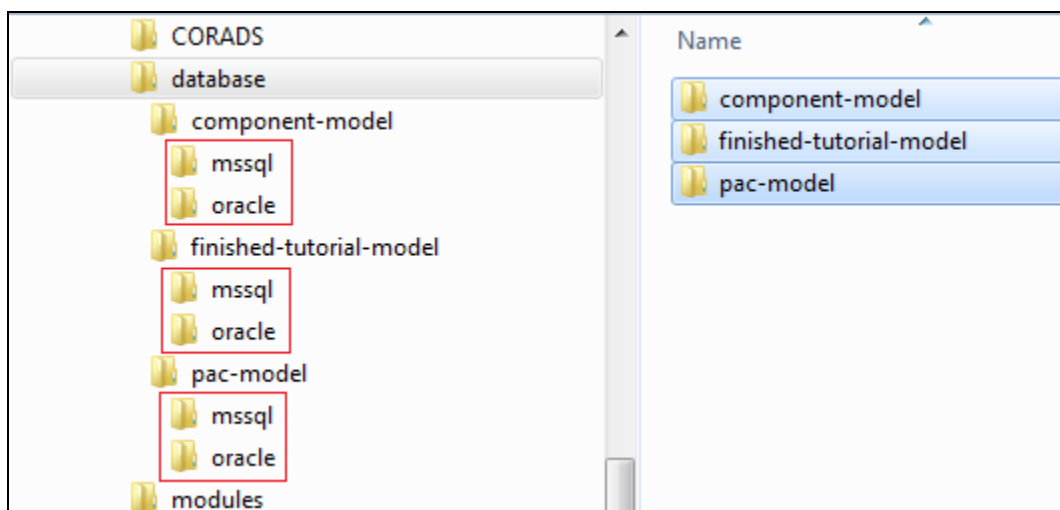
Defect A table that contains a record for each specific defect/bug that is being tracked. This table contains data about each defect, including a description of the problem, severity, priority, to whom the defect is assigned, and so on.

You are going to build a browser-based client using Tomcat and Microsoft SQL Server.

Before You Begin: Environment Setup

For this tutorial, you will be using Microsoft SQL Server for your database system and Tomcat as your application server in your test environment. Before you can complete this tutorial, you must be sure that you have CodeObjects Platform and CORADS installed as described in the **Getting Started** manual.

The image below shows you the database (DB) scripts that cover three areas: the component-model, the finished-tutorial-model, and the pac-model. You can find the database script to create the DB tables for each area at the sub-folder per the database vendor. For example, the component-model\oracle contains the DB scripts to create the necessary tables used by common components or enterprise components.



Refer to the following table to determine which DB scripts to run according to your intended use.

The application scope	SQL Scripts you must run
If you plan to build your own application on top of the CodeObjects common components	Run the SQL scripts under <i>component-model</i> .
To go through the tutorial application	1. Run the SQL scripts under <i>component-model</i>. 2. Run the SQL scripts under <i>finished-tutorial-model</i>.
If you plan to build your own application on top of the CodeObjects Property & Casualty application.	1. Run the SQL scripts under <i>component-model</i>. 2. Run the SQL scripts under <i>pac-model</i>.

Servers

SMTP A working SMTP server for outbound email (you'll use Google Gmail in this tutorial.) You can leverage your existing SMTP server in your network, or you can use any known public internet email servers that support SMTP protocol for the exercise. This SMTP server is not used in the first tutorial, but will be used in a later tutorial.

WebDAV A working WebDAV server for attaching files to the file system. Microsoft Internet Information Service (IIS) is one option. This WebDAV server is not used in the first tutorial, but will be used in a later tutorial.

Although WebDAV configuration is beyond the scope of this tutorial, we offer the following suggestions. A typical Windows desktop or Windows server includes the IIS option, which you can enable in Control Panel. Depending on your Windows version (Windows XP, Windows Vista, Windows 7, Windows Server 2003, Windows Server 2008), you should be able to find detailed configuration information at <http://www.microsoft.com> for enabling WebDAV or at http://www.windowsnetworking.com/articles_tutorials/WebDAV-IIS.html. Be sure that your setup allows you to attach and delete documents, not just read-only access.

Prerequisites

To perform this tutorial, you already should have:

- A basic working knowledge of Eclipse—you should know about workspaces and projects and standard Eclipse interface elements such as views and perspectives.
- Knowledge of your source control system (optional)—we will show you what to check in, but you must know about the location and operation of your own source control system.

What You Will Create

You are going to create a complete application using CORADS, following the model and process outlined in the **Getting Started** manual. You will use CORADS to define the metadata in each of the layers associated with the CodeObjects Platform modeling system:

1. **Data layer** You will import the data table definitions from the database and define metadata entities associated with the data tables.
2. **Services layer** You will define the service that you are going to deploy.
3. **Deployment** You will set up the deployment layer (for the service) and deploy the service to test it.
4. **Client/UI layer** You will build a basic interface to display a client.
5. **Deployment** You will set up security and settings for your application, creating a complete J2EE application, and deploy it to Tomcat for testing in your browser. The J2EE application is a WAR file that contains all the generated artifacts produced through the CodeObjects transformation engine for all the different metadata layers.

User Interface Terminology

Buttons

The Eclipse user interface frequently displays several buttons that do not have universally accepted names. To avoid confusion or the need to show a small graphical image at each mention, we define the names that are used for these buttons throughout these tutorials.

Pick button



Opens a dialog box from which to pick an item or occasionally, a file location.

Pick Property button



Appears in the Property pane at the right edge of a Value field for a property. It opens a dialog box from which to pick a value.

List arrow



Appears in the Property pane at the right edge of a Value field. It opens a list containing a discrete selection of values, including a blank (no value).

Left Arrow



Moves a selection from a right pane to a left pane.

Right Arrow



Moves a selection from a left pane to a right pane.

The images that appear in your user interface may be slightly different from those shown here, depending on your operating system and the configured theme, but you will have no difficulty recognizing them. For the first few instances that a particular button appears in the tutorials, or at times to provide emphasis, a small image will accompany the button name. After a few instances, we will just use the button names without an image; as for example, click the **Pick** button.

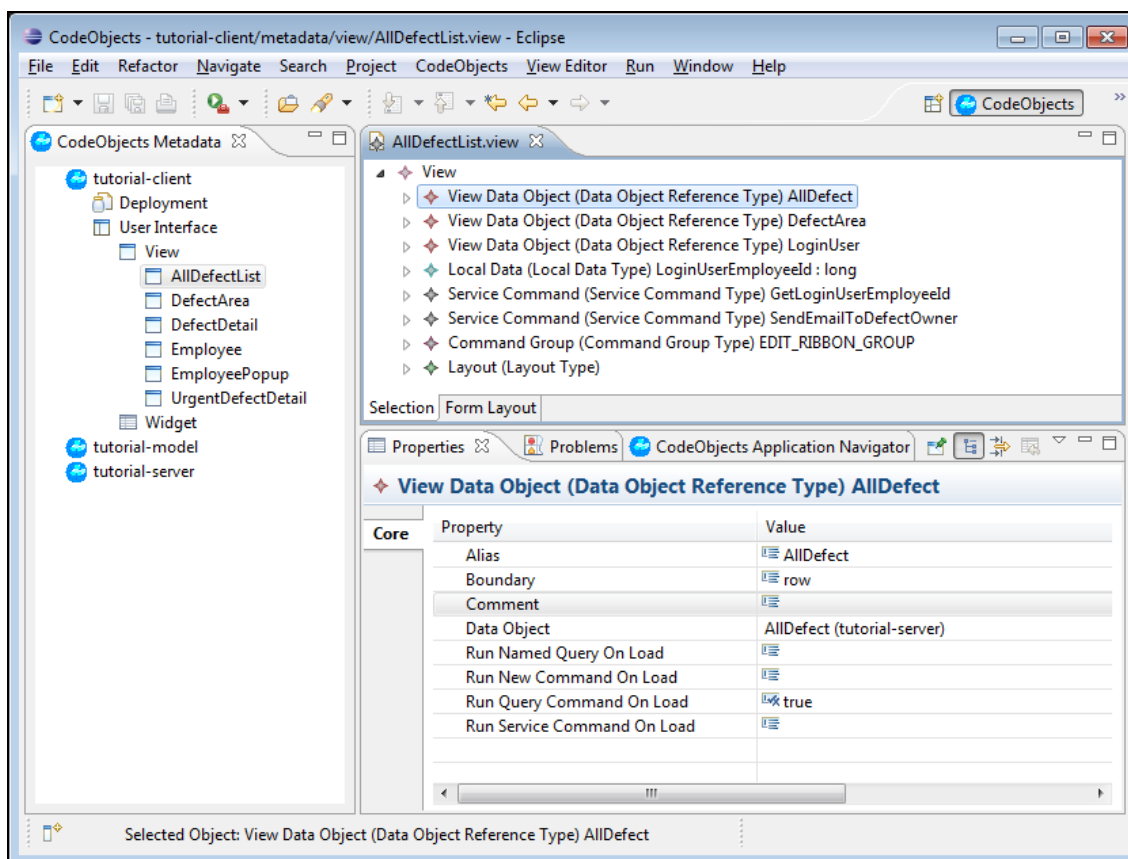
Screen Layout

The Eclipse workbench offers standard and customizable layouts called *perspectives*. The specific layout for CodeObjects application development is called the *CodeObjects Perspective*, which is also used throughout the tutorials. To access the layout for using the Eclipse workbench with the CodeObjects Perspective, or to restore it if it should become necessary:

- From the menu, click **Window** → **Open Perspective** → **Other**.
In the **Open Perspective** box, click **CodeObjects** and **OK**.

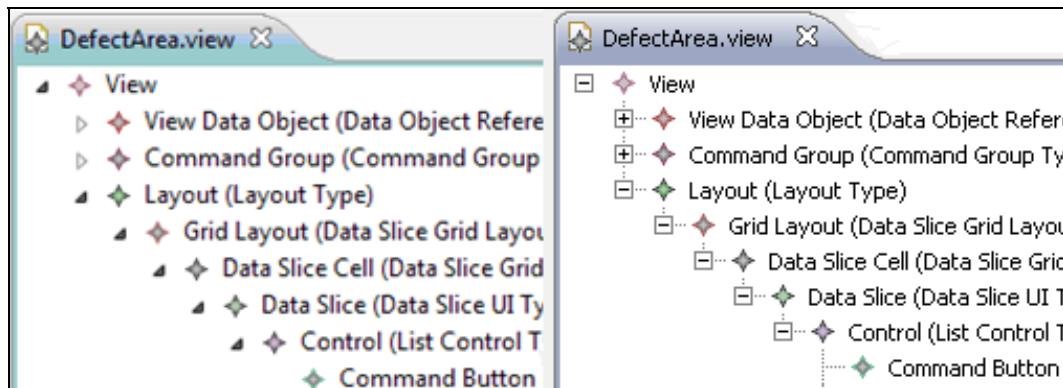
The CodeObjects Perspective, as configured and documented within the tutorials, contains three major work areas, called *panes*, along with the menu that runs along the top.

- Metadata pane** This pane occupies the left side of the workbench and displays a tree hierarchy of the overall metadata structure, in much the same way that a folder tree displays the hierarchical structure of files in a file system. Initially, the pane is empty, but you will populate it as you add metadata to your application. The pane sometimes includes multiple tabs. The primary tab that you will use is the CodeObjects Metadata tab. You will access the tutorial compiled WAR file under the Package Explorer tab and database connections under the Data Source Explorer tab.
- Editor pane** This pane occupies the upper-right side of the workbench and usually displays a tree hierarchy of tables, objects, entities, views, and so on, in a multiple tab layout. Additional tab controls at the bottom of the pane provide convenient access to Form Layout for designing and editing application forms.
- Property pane** This pane occupies the lower-right side of the workbench and displays multiple tabs. You will primarily use the Properties tab that contains the properties (and their values) of the items that appear in the above Editor pane.



Hierarchical Trees and Command Sequences

Hierarchical tree elements are each called nodes, and include control icons that show whether nodes contain sub-nodes. The icons appear as small arrowheads or as plus or minus signs, depending on your operating system and configured theme.



Click a right arrow/plus sign to expand the view of sub-nodes or an angled arrow/minus sign to collapse the view. A node without an arrow/sign has no sub-nodes.

Hierarchical tree structures are often complex and difficult to describe in a text narrative. To describe such a structure, or a hierarchical menu/command sequence, we place the individual elements sequentially in bold text separated by right arrow (→) characters.

For a node example: **Data Table → Data Column (Data Column Type) CO_ID**

For a menu example: Click **File → Save**

Some nodes have especially long names. If there is no possibility of confusion, a long name may be abbreviated and contain an ellipsis (...), or possibly two. For example, the node **Dynamic List of Values Field (Dynamic List of Values Field Type) DefectAreaName** might be represented as **Dynamic List of Values Field...**

Saving Your Work

As you work through the tutorial, you will be reminded to save your work.

To save the currently open component, click **File → Save** or press CTRL+S.

To save *all* of the project components, click **File → Save All**.

These *Save* commands are available only when you have made changes that have not been saved already. Saving changes frequently is a good idea.

TIP

If you find that a step in the tutorial isn't working, try saving your work before proceeding.

Setting up the Application

In this chapter, you are going to set the basic settings in Eclipse, create the modules (Eclipse projects) that you will use for your application, and create the database connection profile.

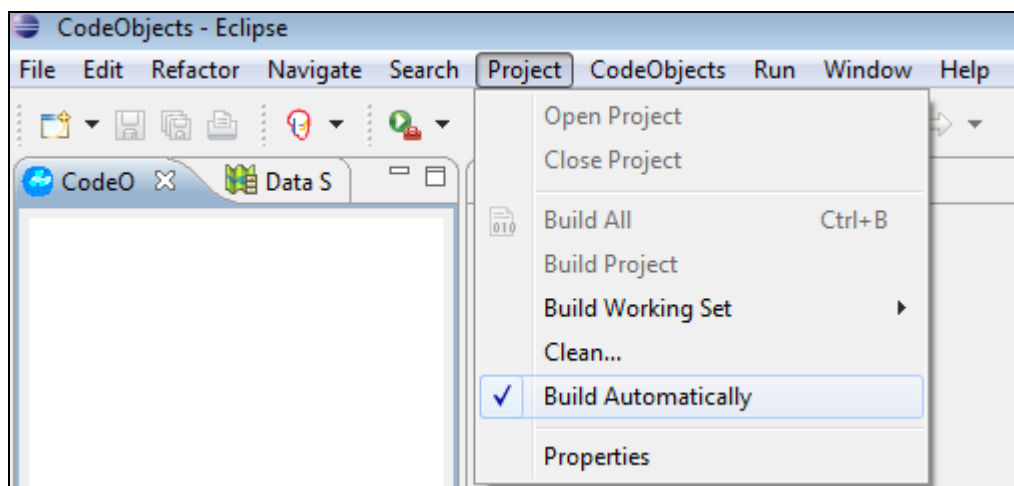
Begin by running Eclipse. Depending on Eclipse's state when you begin this tutorial, you might see a tab labeled *Welcome* and its pane containing *Welcome to Eclipse for RCP and RAP Developers*. Close the tab. Do not be concerned if the layout looks different from that in the image on this page. You will reset to the CodeObjects perspective shortly.

Throughout the tutorial, when we refer to a menu item, we are referring to the Eclipse menu.

Enable Automatic Builds

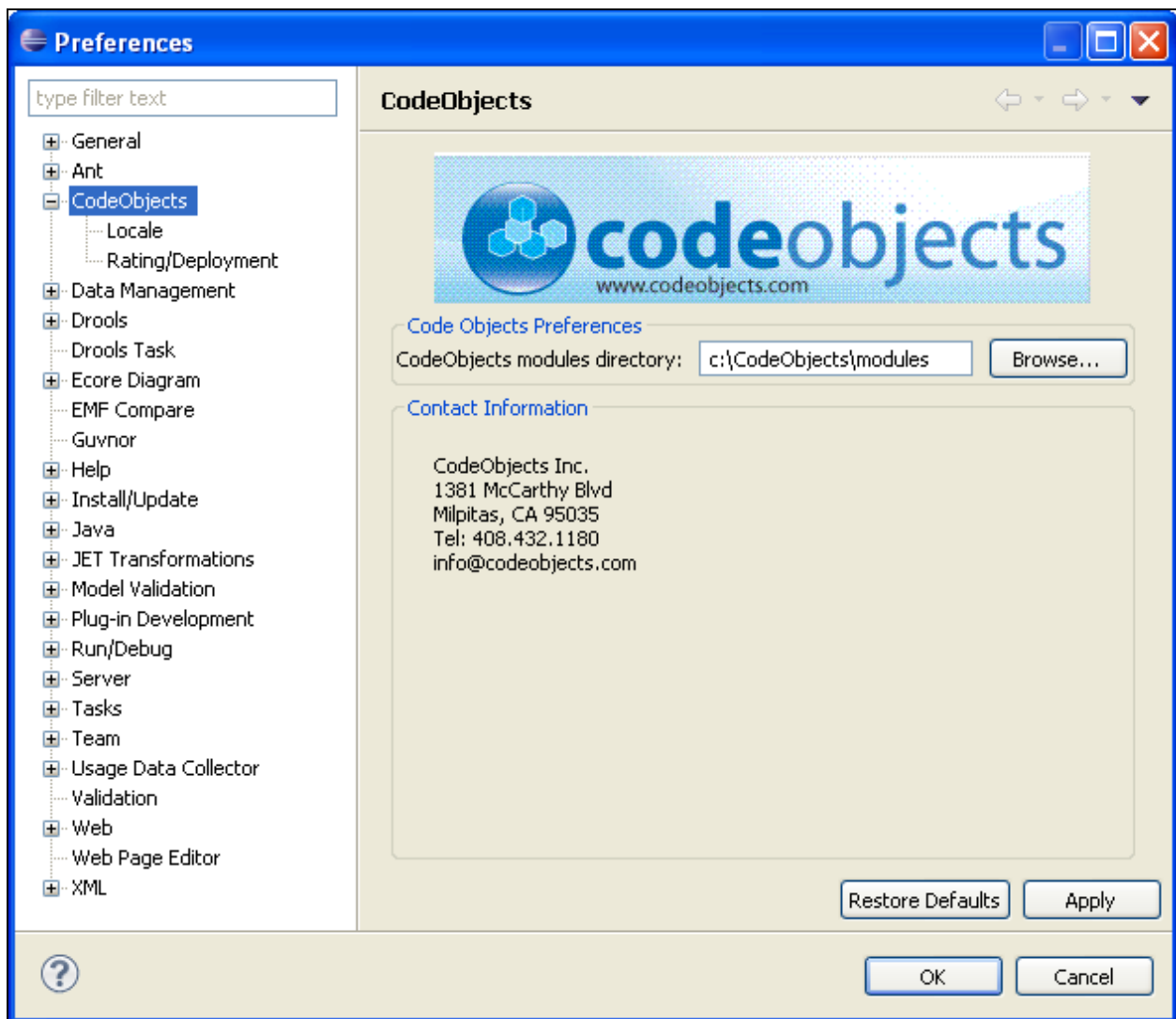
Set the automatic build option so that CORADS will automatically build (compile your metadata) whenever you save your changes. This will speed up your development process because error indicators will appear if there are any compilation errors, and the right artifacts will be ready when you want to test/deploy.

Click the **Project** menu. A check mark should indicate that **Build Automatically** is enabled. If it is not enabled, click **Build Automatically**.



Set the Modules Directory

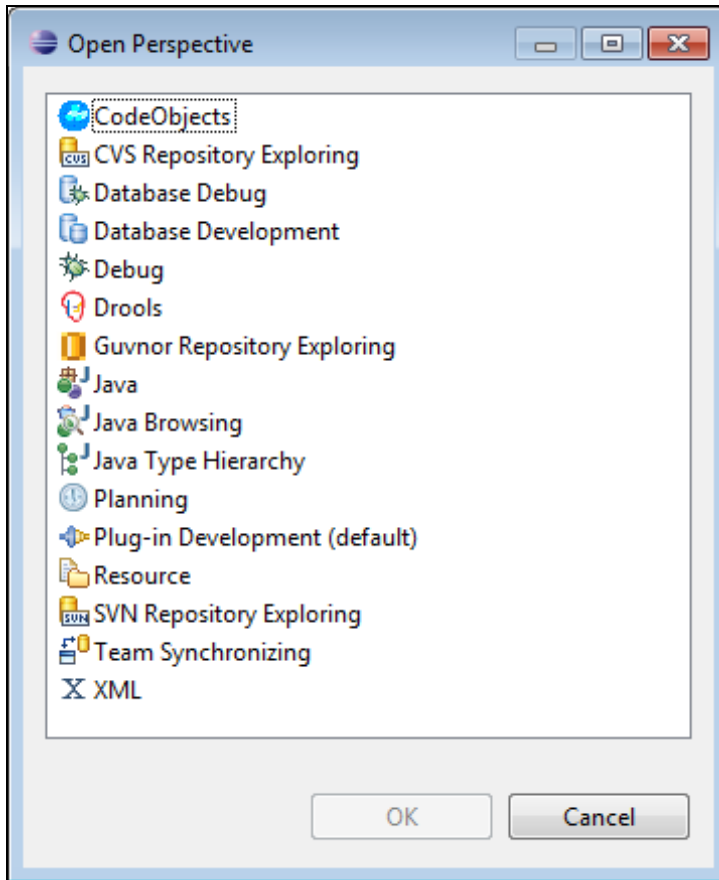
From the menu, click **Window** → **Preferences** to open the **Preferences** dialog box, and then in the list at the left, click **CodeObjects**. For the **CodeObjects modules directory**, you could enter the path (if you know it) directly in the text box field, but we recommend that you click the **Browse** button and navigate to the modules directory, under the directory where you installed CodeObjects Platform. Click **OK** as necessary to close dialog boxes.



Open CodeObjects Perspective

From the menu:

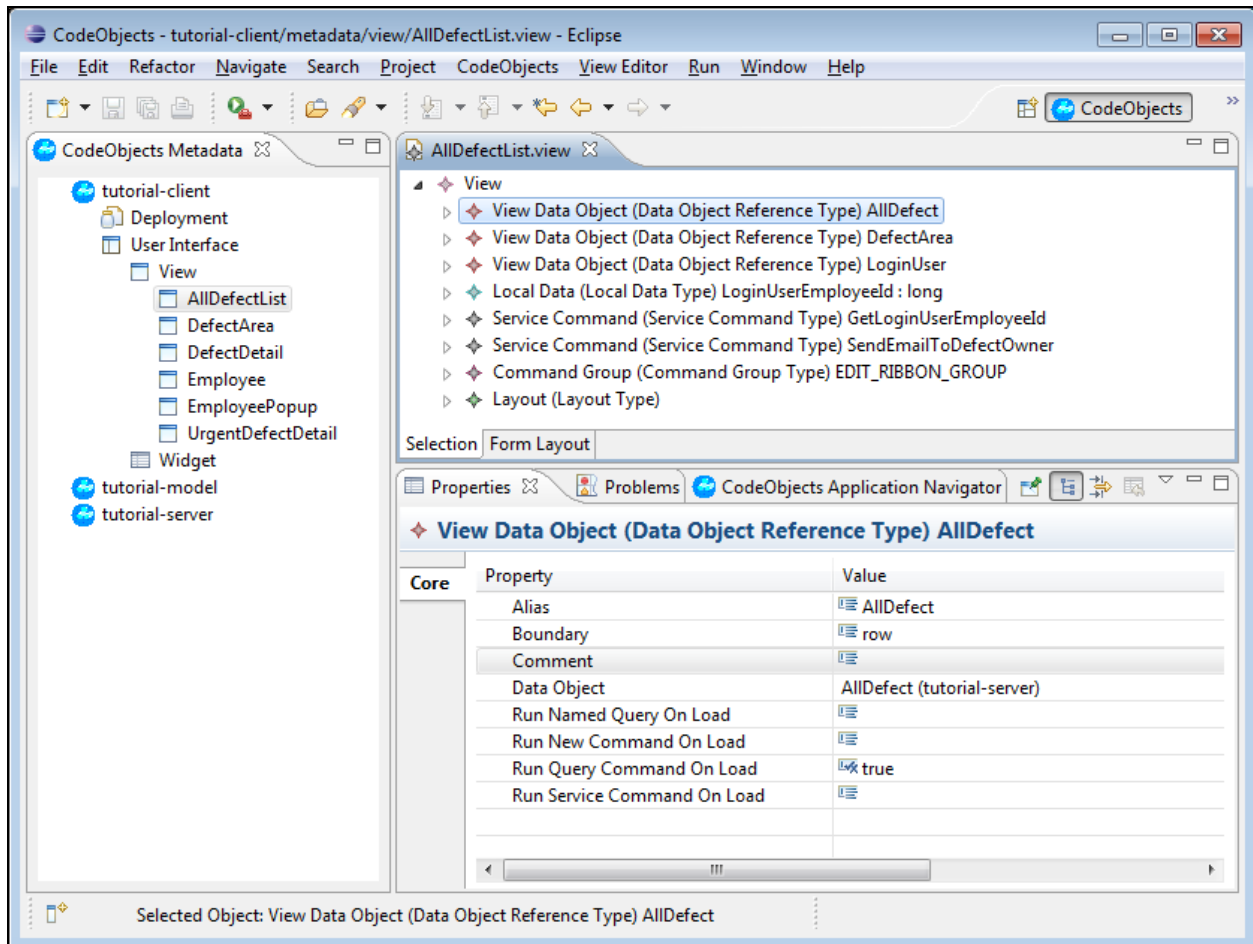
1. Click **Window** → **Open Perspective** → **Other**.
2. Click **CodeObjects**, and then click **OK**.



Setting up the Application

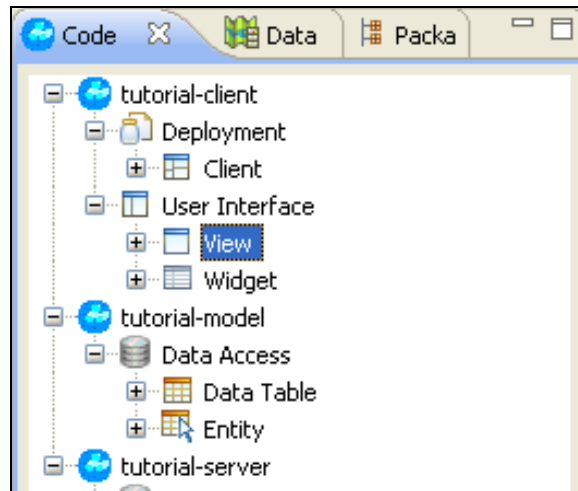
The Metadata Pane

The *CodeObjects Metadata pane* (usually called simply the *Metadata pane*), located at the left of the Eclipse window, displays the high level metadata items as you create them. Metadata items are displayed as a tree hierarchy, and you can click through the tree to review the metadata at any time.



Setting up the Application

At this beginning stage, the Metadata pane is empty, but as you work through this tutorial, you will gradually populate the pane with metadata. Eventually, the pane will appear something like this:



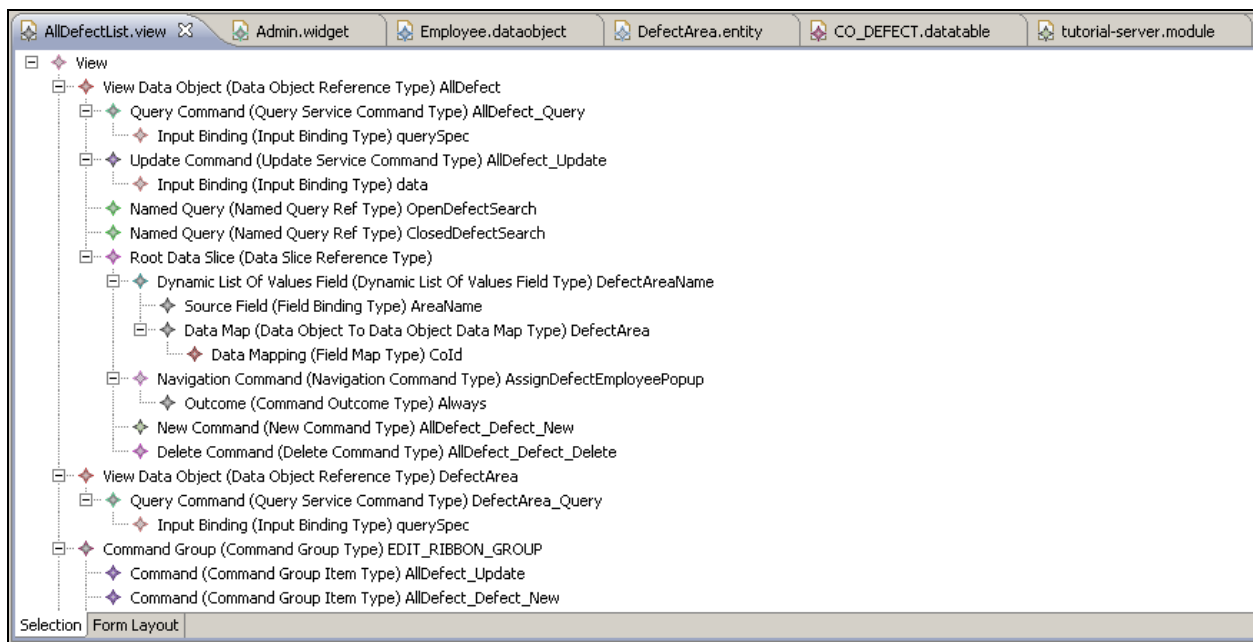
In this tutorial, you are going to create the metadata for a basic application. At each section, it would be a good idea to pause after creating the metadata to review how the newly created items appear in the *CodeObjects Metadata pane*. You can look at each item's properties to confirm that they were created correctly. In the Metadata pane, double-click the item to display the detailed components in the Editor pane.

Setting up the Application

The Editor Pane

The Editor pane, located in the upper right of the Eclipse window, contains all the detailed components from which you will build and edit your application. The CodeObjects system builds many of the nodes for you. You will also create nodes yourself, mainly by adding new child nodes to existing ones as refinements or added features for your application.

Each tab in the Editor pane corresponds to an item in the Metadata pane. The tabs in the following image show some the many kinds of data elements an application might include.



The small *Selection* tab at the bottom of the pane indicates that the pane is intended for node selection.

Setting up the Application

Form Layout

The *Form Layout* tab provides a quasi-WYSIWYG interface designed for laying out views for your application's user interface.

The screenshot shows a software development environment with a 'Form Layout' tab selected. The form is designed for a defect management system. It features several input fields and dropdown menus arranged in a grid-like structure. The fields are labeled with their respective data types in parentheses. The form is divided into sections by horizontal lines. The bottom of the window shows a 'Selection' and 'Form Layout' tab.

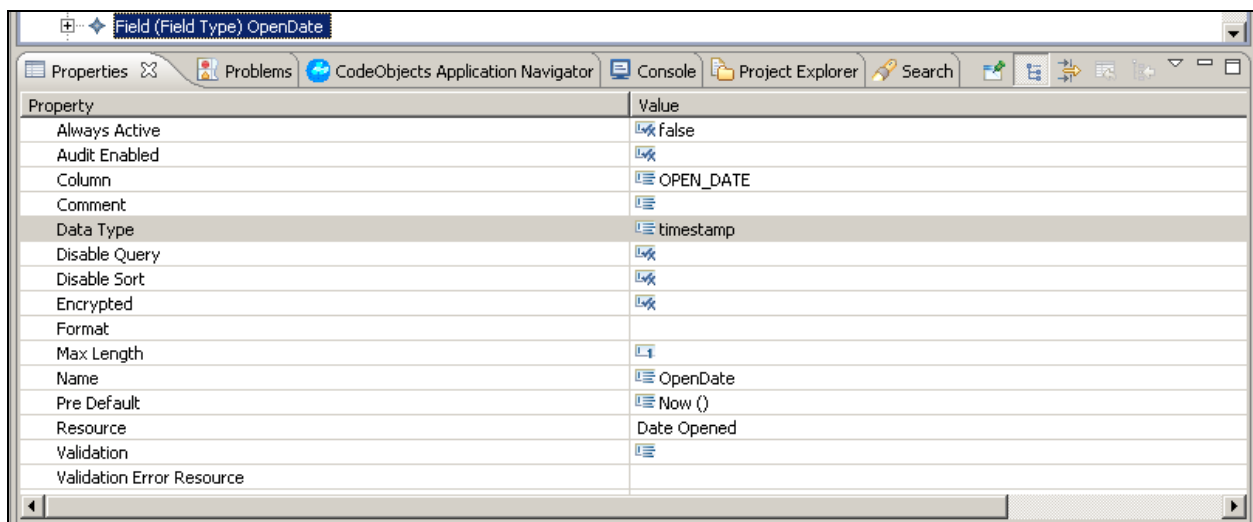
Date Opened:	(OpenDate)	Defect Area:	(DefectAreaName)	(Severity)	(Severity)
Due Date:	(DueDate)	Owned By:	(OwnedByEmployeeNa...)	(Priority)	(Priority)
Closed Date:	(ClosedDate)	Closed By:	(ClosedByEmployeeName)	(Status)	(Status)
Assigned Date:	(AssignedDate)	Created By:	(CoCreatedBy)	Sub Status:	(SubStatus)
(Description)	(Description)				
Customer Impact:	(ClientImpact)				
(Resolution)	(Resolution)				

Click a component in the Editor pane to display its properties in the Property pane.

The Property Pane

The Property pane, located in the lower right of the Eclipse window, contains properties of the selected node in the Editor pane. The CodeObjects system specifies some properties for you, and you will edit others as necessary. The properties appear as two columns, with each Property on the left and the corresponding Value on the right. Depending on the property, you specify its value by typing it in the value field, choosing it from a drop-down list, or clicking a pick button to open a box containing choices.

Although you will work primarily under the Property tab, you will occasionally use one of the other available tabs that appear in the following image.



Create a Folder for Your Application

Create a new folder to store your application files for the defect tracker application that you will build in this tutorial. Throughout this tutorial, we will assume that the application files are located at:

<CodeObjectsInstallRoot>\modules\tutorial

Create Modules for the Tutorial Application

A module is a container for grouping metadata and programming extensions. (If you've used Eclipse before, you may consider a CodeObjects module as an Eclipse *project*.)

For the defect tracker application, you will create three modules:

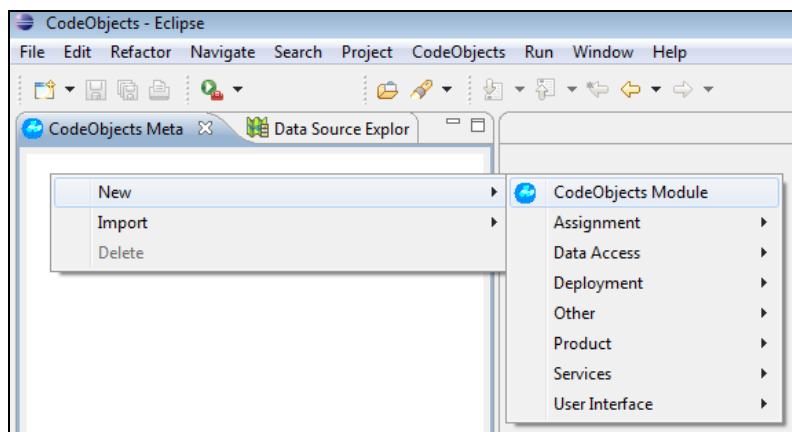
Name	Description	Root Path	Root Package
tutorial-model	Basic metadata (e.g., Data Table, Entity) which represents the base data model of the entire tutorial application.	<CodeObjectsInstallRoot>\modules\tutorial\tutorial-model	com.app.tutorial.model
tutorial-server	Services layer metadata (e.g., Service, Data Object) as well as the service deployment.	<CodeObjectsInstallRoot>\modules\tutorial\tutorial-server	com.app.tutorial.server
tutorial-client	Client layer metadata (e.g., View, Widget) as well as the client deployment.	<CodeObjectsInstallRoot>\modules\tutorial\tutorial-client	com.app.tutorial.client

Table 1 Tutorial modules

Each module must be stored in a separate folder. Later sections of this tutorial will discuss more about which files are created.

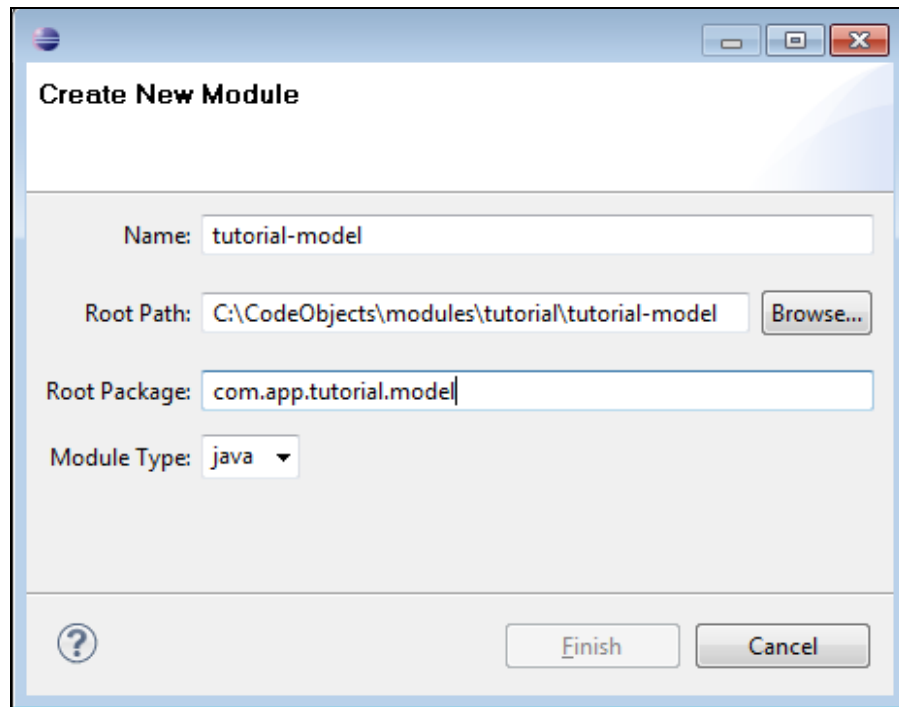
To create a module:

1. Right-click anywhere within the **CodeObjects Metadata** panel, and from the context menu, choose **New** → **CodeObjects Module**.



Setting up the Application

2. The **Create New Module** dialog opens. Fill in the correct values (refer to the table using the values listed in the Tutorial modules table on page 15.)
3. Click the **Browse** button to select the folder that you created for your application. In the **Browse For Folder** dialog box, click the **Make New Folder** button to create a new sub-folder for your module, and then click **OK** as necessary.



NOTE

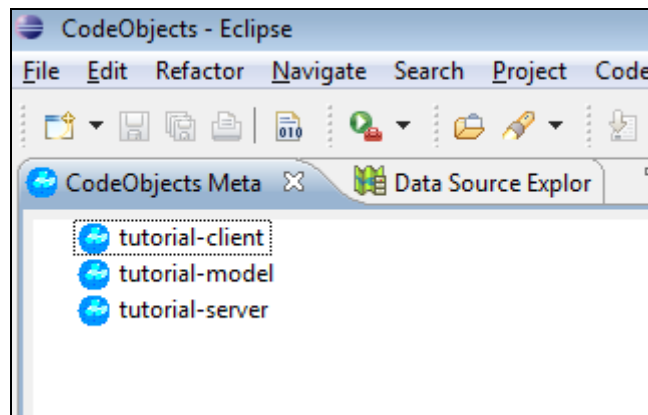
You must create a new root path by using the Browse button. If you attempt to create the path directly by simply entering text, the Create New Module operation will fail because the path does not exist.

4. Click **Finish** to complete the module creation.
5. In the same manner, create the other two modules for the defect tracking application. Instead of **tutorial-model** (and **tutorial.model**), use **tutorial-client** (and **tutorial.client**) for the second one, and **tutorial-server** (and **tutorial.server**) for the third one.

After you have created all three modules, they should appear under the **CodeObjects Metadata** tab.

Setting up the Application

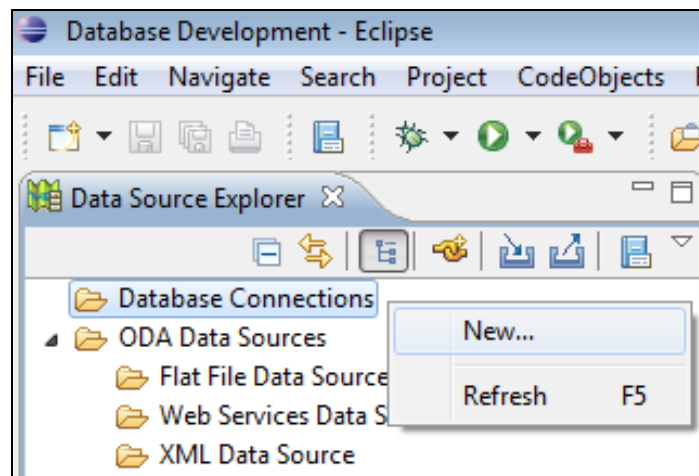
The following image shows the three tutorial modules the **CodeObjects Metadata** tab.



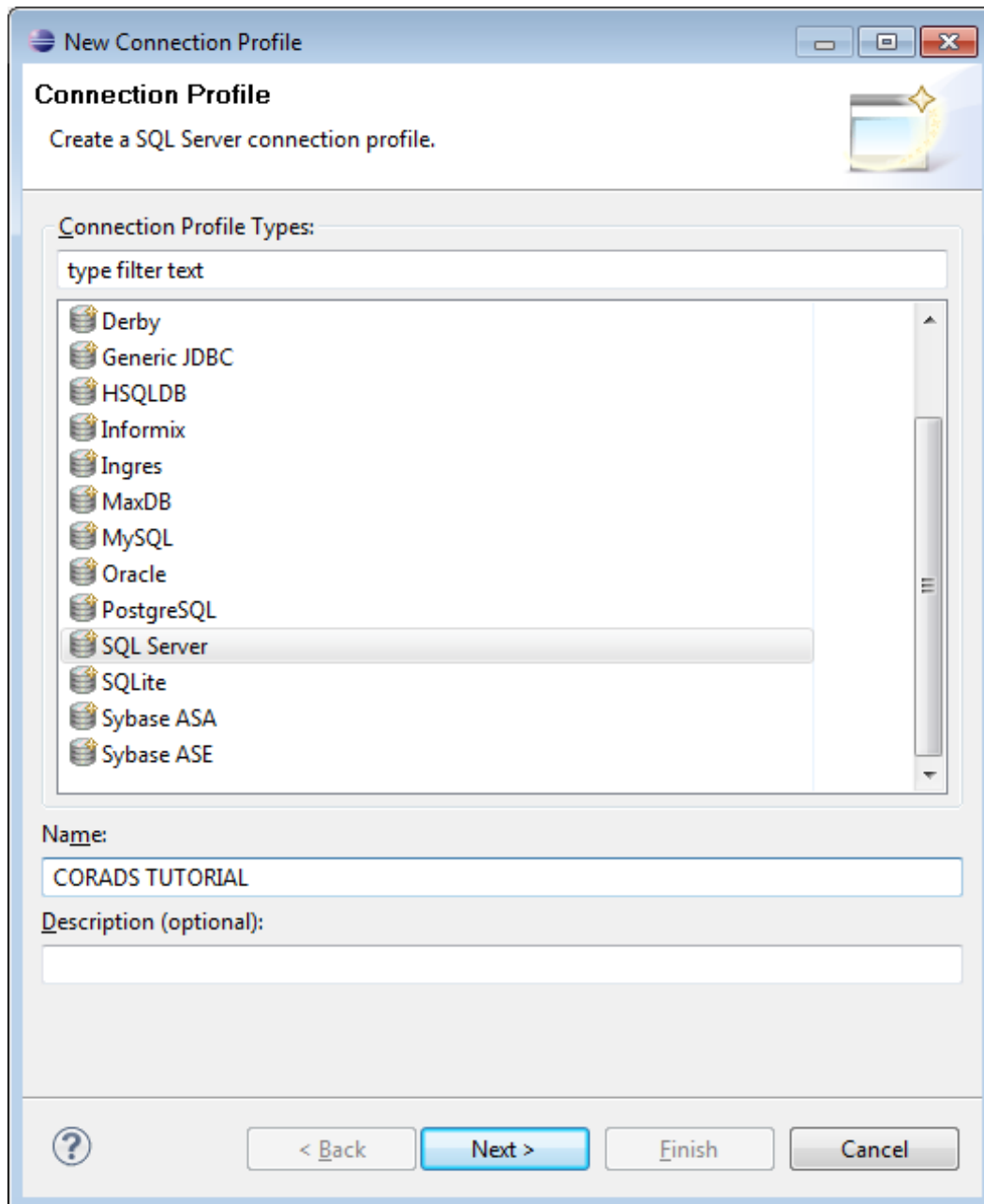
Connect to the Database (Create an Eclipse Connection Profile)

Create an Eclipse connection profile so that you can connect to your database.

1. From the menu, click **Window** → **Open Perspective** → **Other** → **Database Development**, and then click **OK**.
2. Under the **Data Source Explorer** tab, right-click **Database Connections** and choose **New**.

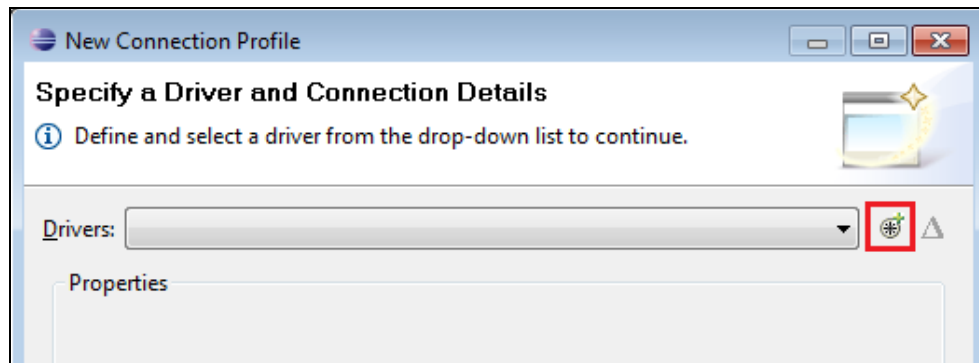


3. In the **New Connection Profile** dialog box, choose **SQL Server** and under **Name** enter **CORADS TUTORIAL**, and then click **Next**. Note that the Name really can be any name you choose. We recommend that you choose a name that matches your database name.



Setting up the Application

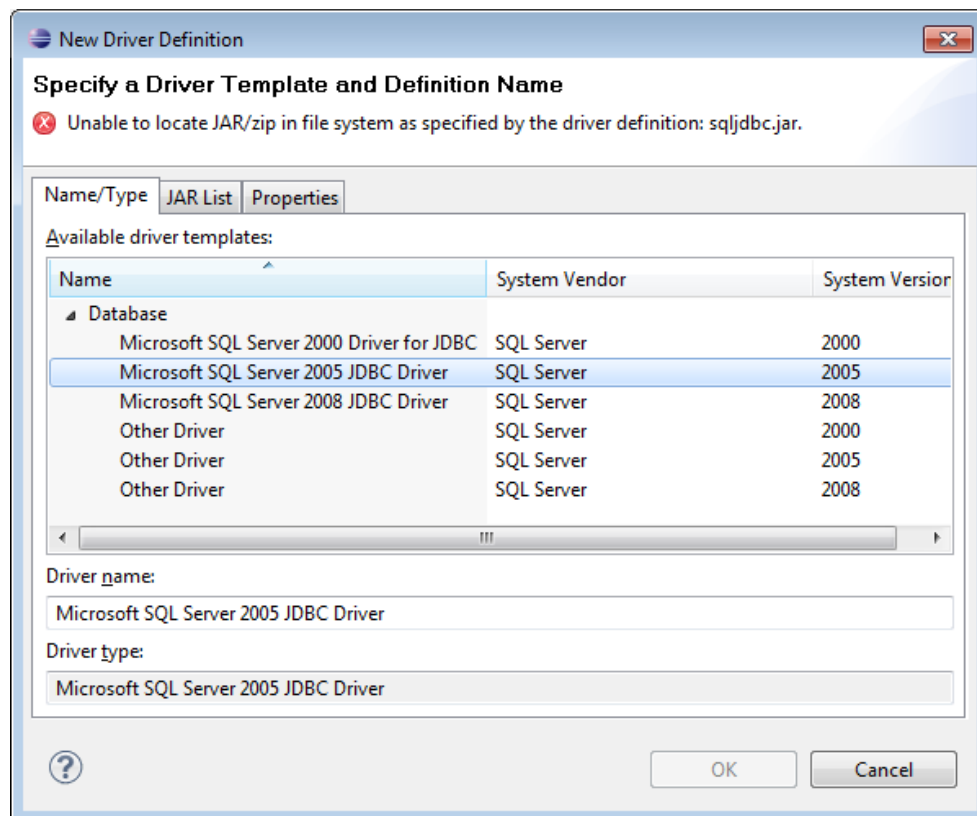
4. From the **Drivers** list select a driver, or if no driver is available, click the **New Driver Definition** button  to open the **New Driver Definition** dialog box.



Create a New Driver Definition

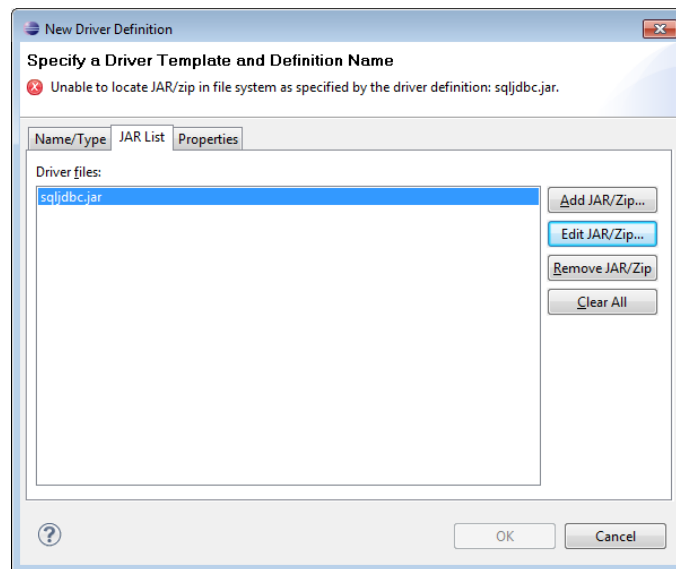
To create a new driver definition:

1. Under the **Name/Type** tab, from the list of available driver templates choose **Microsoft SQL Server 2005 JDBC Driver** or **Microsoft SQL Server 2008 JDBC Driver**. This tutorial will work with either SQL Server 2005 or 2008, and the process is exactly the same.



Setting up the Application

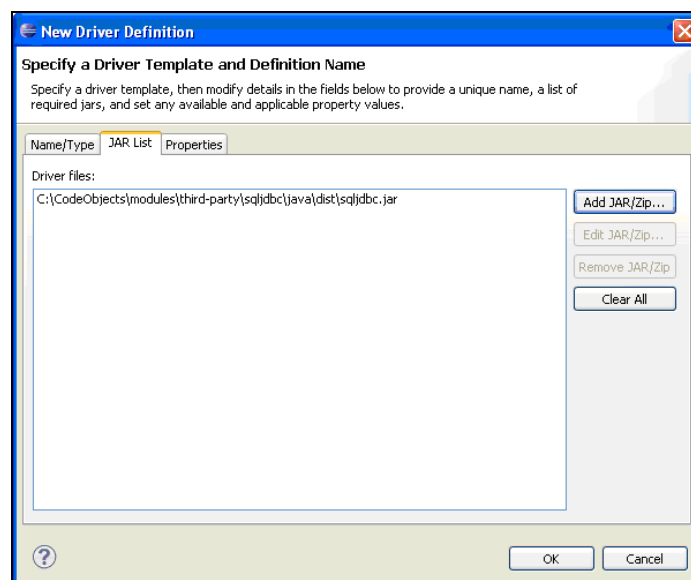
2. Click the **JAR List** tab. If this is the first time you have created a connection profile, the dialog box will show the error *Unable to locate JAR/zip in file system as specified by the driver definition: sqljdbc.jar*. Click **Clear All**, and then click **Add JAR/Zip**.



3. Navigate to the CORADS folder and click the file **sqljdbc.jar**

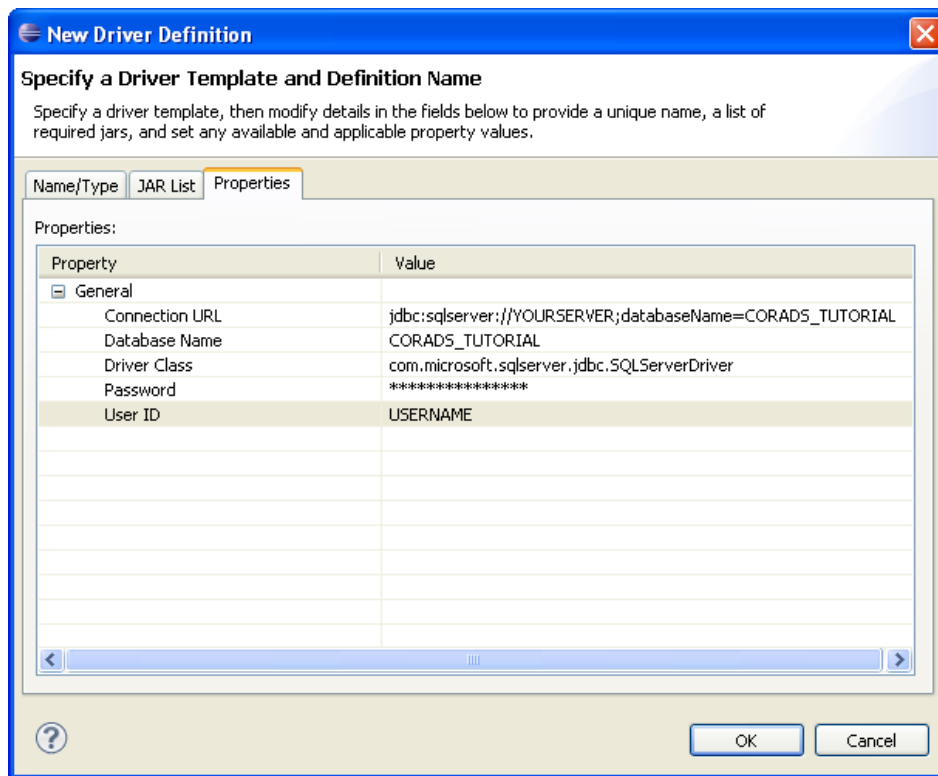
**<CodeObjectsInstallRoot>\modules\third-party\
sqljdbc\java\dist\sqljdbc.jar**

*Do not click **OK** yet.* You must make additional settings elsewhere in this dialog box.
Note that this is a SQL JDBC driver that we have tested with SQL Server 2005 and 2008,
and provided for your convenience.



Setting up the Application

- Click the **Properties** tab and set the **Database Name**, a **Password**, and a **User ID**. These should be based on the server configuration at your location. Consult with your IT staff for the correct database server and database user name.



CORADS currently certifies using SQL Server Authentication. In case your SQL Server uses Windows Authentication, configure it to use SQL Server Authentication, as above.

- Click **OK**.

Test the Connection

After you have defined a driver:

1. The **New Connection Profile** dialog box displays the newly created driver. Be sure that your Database name, machine name (Host), User name, and Password are set correctly.

New Connection Profile

Specify a Driver and Connection Details

Select a driver from the drop-down and provide login details for the connection.

Drivers: Microsoft SQL Server 2008 JDBC Driver

Properties

Database: CORADS_TUTORIAL

Host: YOURSERVER

Port number:

☐ Use integrated authentication

User name: sa

Password:

☒ Save password

Connection URL: jdbc:sqlserver://YOURSERVER;databaseName=CORADS_TUTORIAL

☒ Connect when the wizard completes

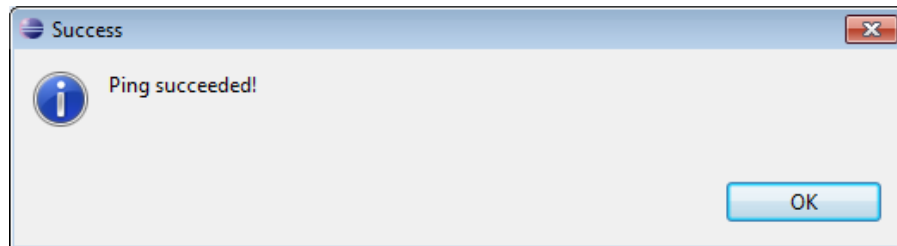
☐ Connect every time the workbench is started

Test Connection

? < Back Next > Finish Cancel

Setting up the Application

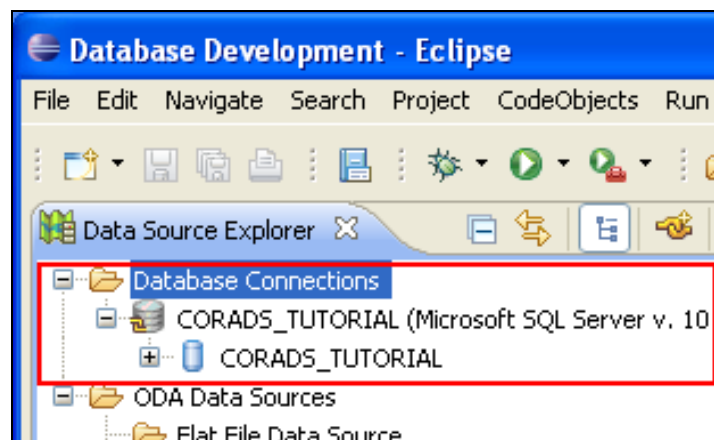
2. Click the **Test Connection** button to make sure you see a **Ping succeeded!** message to indicate a successful connection. If so, click **OK** to close the test box.



WARNING

If you do not receive the **Ping succeeded!** message, Eclipse was not able to connect to your database. You must go back and set your database parameters correctly before proceeding. The remainder of the tutorial relies on having an active database connection available, and you must complete this test successfully before proceeding.

3. Click **Finish** to complete the connection definition. The Eclipse window updates to show the connection under the **Data Source Explorer** tab:



To verify that your database connectivity is working, you can also expand the database to explore the database detail.

Defining the Data Layer

In this chapter, you're going to define the metadata associated with the data layer of the CodeObjects Platform model.

You will do the following steps:

1. Import data table definitions from the database to which you connected in the previous step.
2. Define metadata **entities** for each data table—entities are used to define higher-level properties of your data, such as validation rules and access rules.
3. Use the entities to define additional characteristics of the data and create relationships among data.
4. Define a **data object** based on your entities that will be used to generate the service. A Data Object is composed of one or more entities, and the Data Object makes the entities accessible through SOA service or through programming extensions.

You will use the **tutorial-model** module that you created in the previous chapter to contain the metadata, and you will use data that is defined in the **CORADS_TUTORIAL** database in SQL Server.

High Level Metadata in the Data Layer

The high level metadata objects that you will create in this chapter will appear in the **CodeObjects Metadata** pane as you create them. The high level metadata items that you will create are:

- **Data Table** – analogous to physical database tables, you will create one data table metadata item for each table that you will be using (in this case, two data tables.)
- **Entity** – the logical representation of a data table; in this case, you will set up one entity for each of your tables, but you can create multiple entities for a table if you

want; the entity is a flexible way to define non-physical attributes of data such as validation rules and read-only logic.

- **Data Object** – specifies which entity or entities are to be used for access through a data service, and which fields in the entity are to be accessible. You can think of this as your working data set in your data service.

Displaying the Metadata

This issue sometimes presents itself to tutorial users at this point.

Occasionally, the CodeObjects Metadata tab (or another desired tab) may not be visible when you want to view or use its contents. You might perform an action that disturbs the Eclipse layout, or expands a single pane to fill the screen.

To restore the standard Eclipse layout with the CodeObjects perspective:

- From the menu, click **Window** → **Open Perspective** → **Other**.
In the **Open Perspective** box, click **CodeObjects** and **OK**.

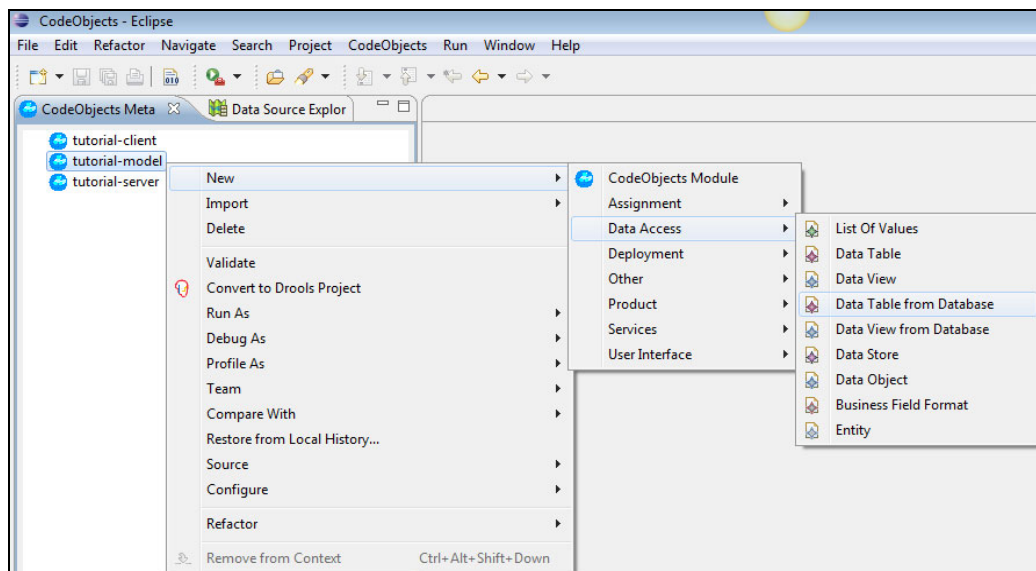
To display the **CodeObjects Metadata** tab
(or the **Package Explorer** or **Data Source Explorer** tab):

- From the menu, click **Window** → **Show View** → **CodeObjects Metadata**
(or **Package Explorer** or **Data Source Explorer**.)

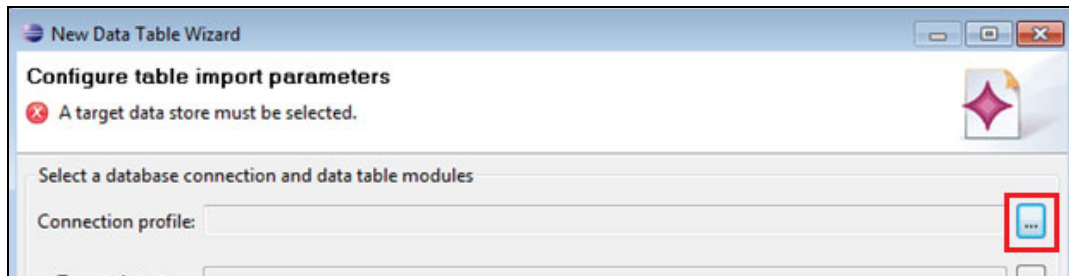
Create a Data Table by Importing from the Database

To import a table definition from the database and create metadata for it:

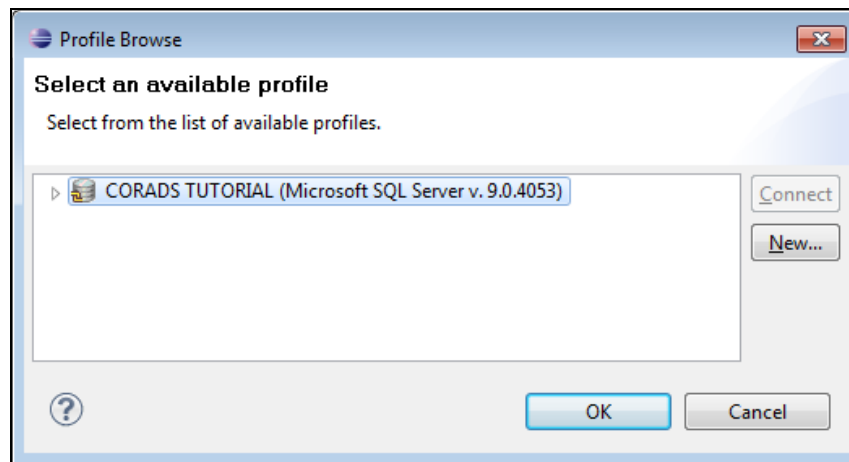
1. In the Metadata pane, right-click the **tutorial-model** module and choose **New** → **Data Access** → **Data Table from Database**.



- At the right side of **Connection profile**, click the **Pick** button .

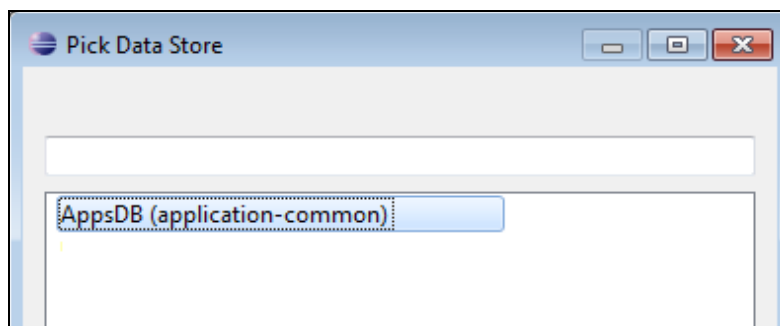


- Select the connection profile you created earlier. If your connection profile is active and is already connected to the database, the **Connect** button will be unavailable. If the **Connect** button is available, click it to connect to the database, and then click **OK**.



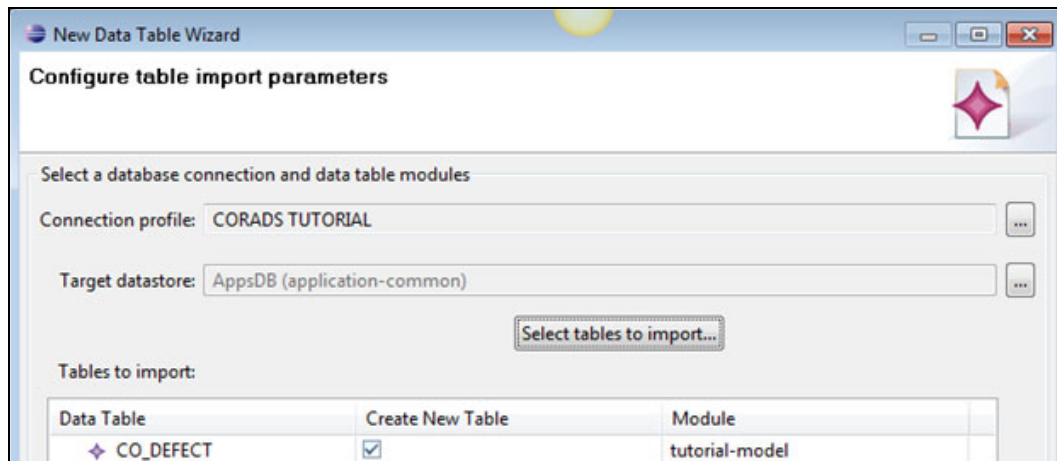
- Adjacent to the **Target datastore** field, click the **Pick** button , and then click **AppsDB** and **OK**.

Note that a Data Store is a high-level logical data collection. **AppsDB** is a pre-defined data store for relational databases.

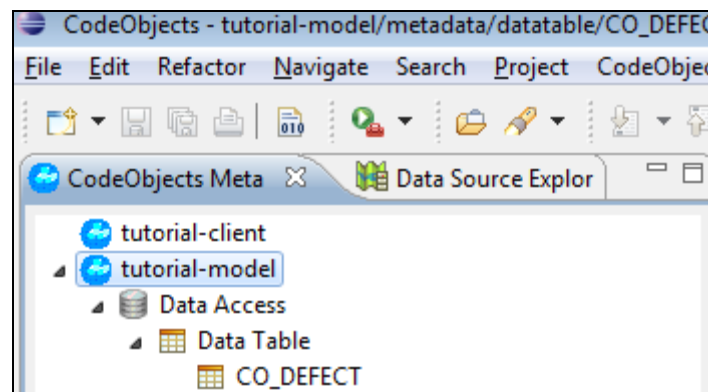


Defining the Data Layer

5. Click the **Select tables to import** button to display all of the data tables in the database. Click the **CO_DEFECT** table. (You will only import a single table for now.)
6. Click the **Right Arrow** to move the name from the **Available** pane to the **Selected** pane.
7. Click **OK**.
8. The **New Data Table Wizard** displays the selected table. Click **Finish**.



9. The imported data table appears in the **CodeObjects Metadata** pane. Expand the tree if necessary to see it.



Double Checking your Results

You have just created your first actual metadata in CORADS. Let's review what CORADS produced and how to verify that everything worked correctly.

The easiest way to verify your work is by examining the **CodeObjects Metadata** pane. The **CodeObjects Metadata** pane displays the **Data Table** heading and its metadata entry.

Double-click the **CO_DEFECT** node to display its metadata in the Editor pane to confirm that the entry was created correctly.

Review your computer's tutorial files folder to see what was actually created. The **tutorial-model** module is stored in **<CodeObjectsInstallRoot>\tutorial\tutorial-model**

The subfolder **metadata** contains the folder **datatable** folder, which in turn contains the data table file that you just imported. The file name is of the form: **<name of metadata item>.<metadata type>**

So, you should see the file **CO_DEFECT.datatable**

Metadata files are XML files that you can edit with any standard text editor. Normally, you manage the metadata through CORADS, rather than by directly editing the files, and you wouldn't refer to these files often. However, to check them in to your source control system, you must know where they reside. (We will cover more about that topic later.)

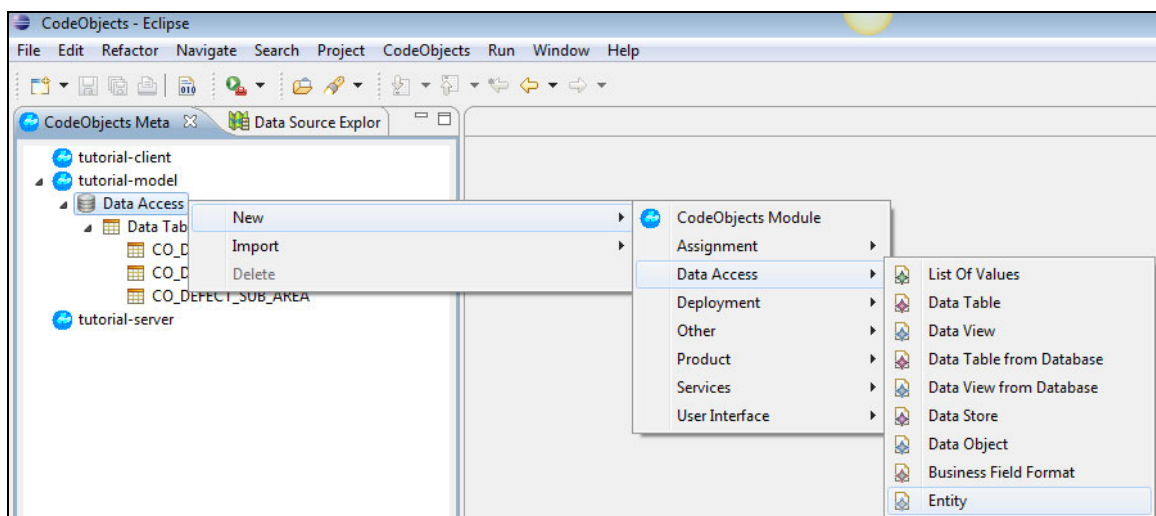
Create an Entity for the Data Table

In this section, you will create a metadata entity called *Defect* for the data table that you imported.

Name	Module	Entity Type	Data Table
Defect	tutorial-model	Simple Entity based on Data Table or Data View	CO_DEFECT

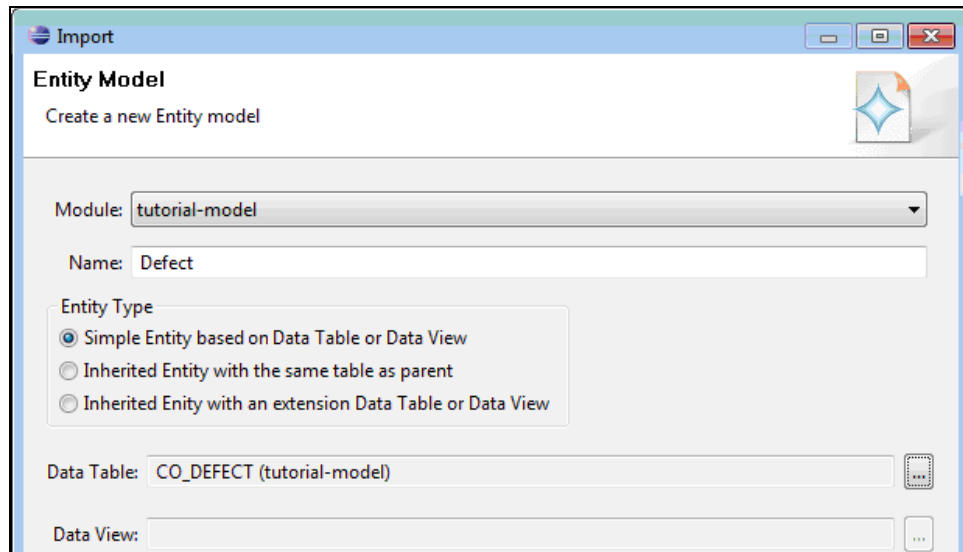
To create the **Defect** entity:

1. In the Metadata pane, right-click the module **tutorial-model** module and choose **New** → **Data Access** → **Entity**.



Defining the Data Layer

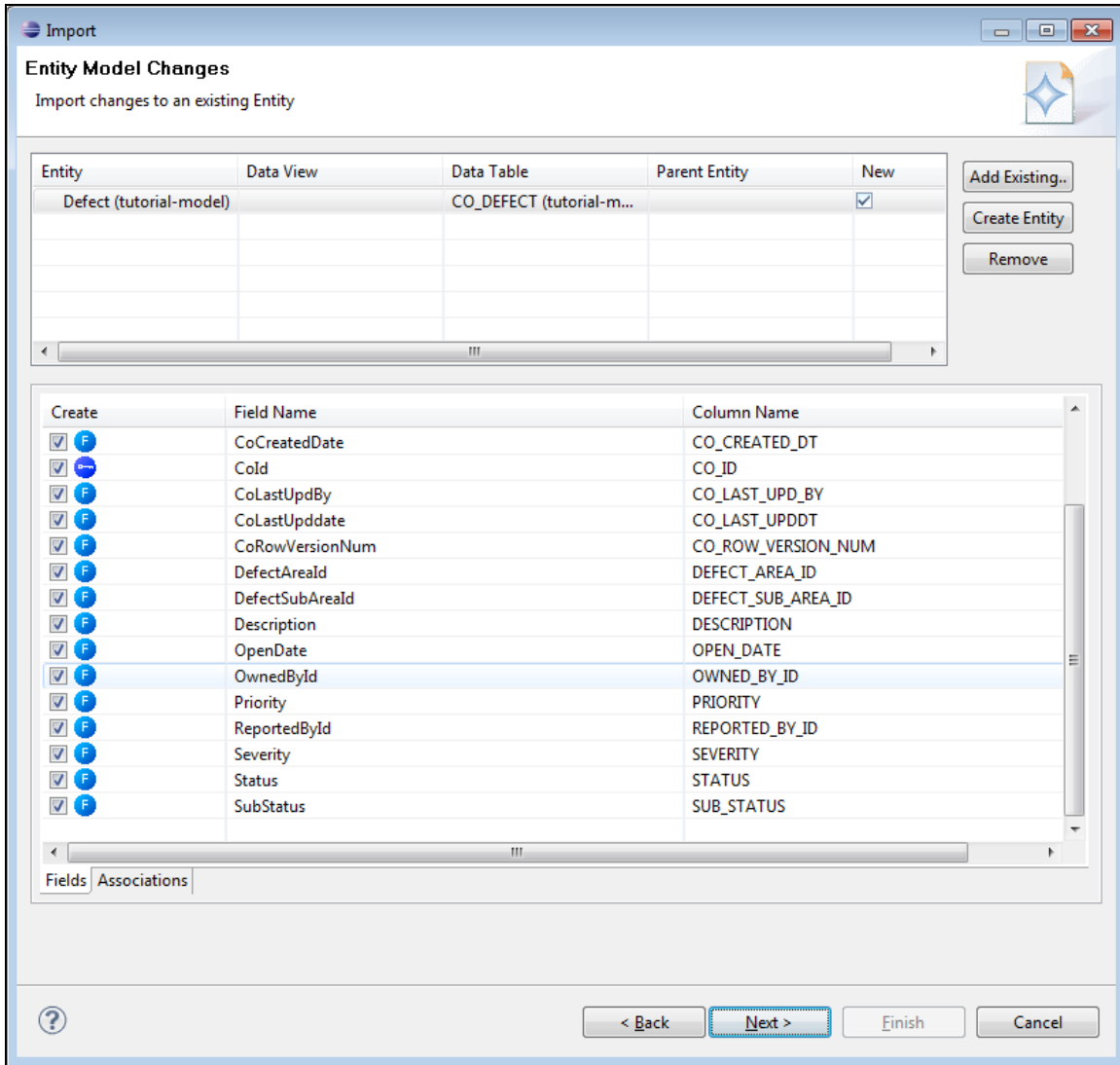
2. From the **Module** list, choose **tutorial-model**. Set the **Name** to **Defect**, and for **Entity Type** click the option button for **Simple Entity based on Data Table or Data View**.
3. At the right side of **Data Table**, click the **Pick** button. In the **Pick Data Table** box start typing **CO_Defect**, choose **CO_Defect (tutorial-model)**, and click **OK**.
4. Click **Next**.



The screenshot shows a dialog box titled "Import" with a sub-header "Entity Model" and the instruction "Create a new Entity model". The dialog contains the following fields and options:

- Module:** A dropdown menu with "tutorial-model" selected.
- Name:** A text input field containing "Defect".
- Entity Type:** A group box containing three radio buttons:
 - ☒ Simple Entity based on Data Table or Data View
 - ☐ Inherited Entity with the same table as parent
 - ☐ Inherited Entity with an extension Data Table or Data View
- Data Table:** A text input field containing "CO_DEFECT (tutorial-model)" and a "Pick" button (represented by a small icon with three dots) to its right.
- Data View:** An empty text input field and a "Pick" button (represented by a small icon with three dots) to its right.

5. Notice the Field and Column Names. Click **Next** to accept the default values.



The screenshot shows the 'Import' dialog box with the 'Entity Model Changes' tab selected. The dialog is titled 'Import' and has a subtitle 'Import changes to an existing Entity'. It contains a table with columns: Entity, Data View, Data Table, Parent Entity, and New. The first row shows 'Defect (tutorial-model)' in the Entity column, 'CO_DEFECT (tutorial-m...' in the Data Table column, and a checked box in the New column. To the right of the table are buttons: 'Add Existing..', 'Create Entity', and 'Remove'. Below the table is a scrollable list of fields to be imported. The list has three columns: 'Create', 'Field Name', and 'Column Name'. Each row has a checked box in the 'Create' column, a field name in the 'Field Name' column, and a column name in the 'Column Name' column. The fields are: CoCreatedDate, Cold, CoLastUpdBy, CoLastUpddate, CoRowVersionNum, DefectAreald, DefectSubAreald, Description, OpenDate, OwnedById, Priority, ReportedById, Severity, Status, and SubStatus. The 'OwnedById' row is currently selected. At the bottom of the dialog are buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted.

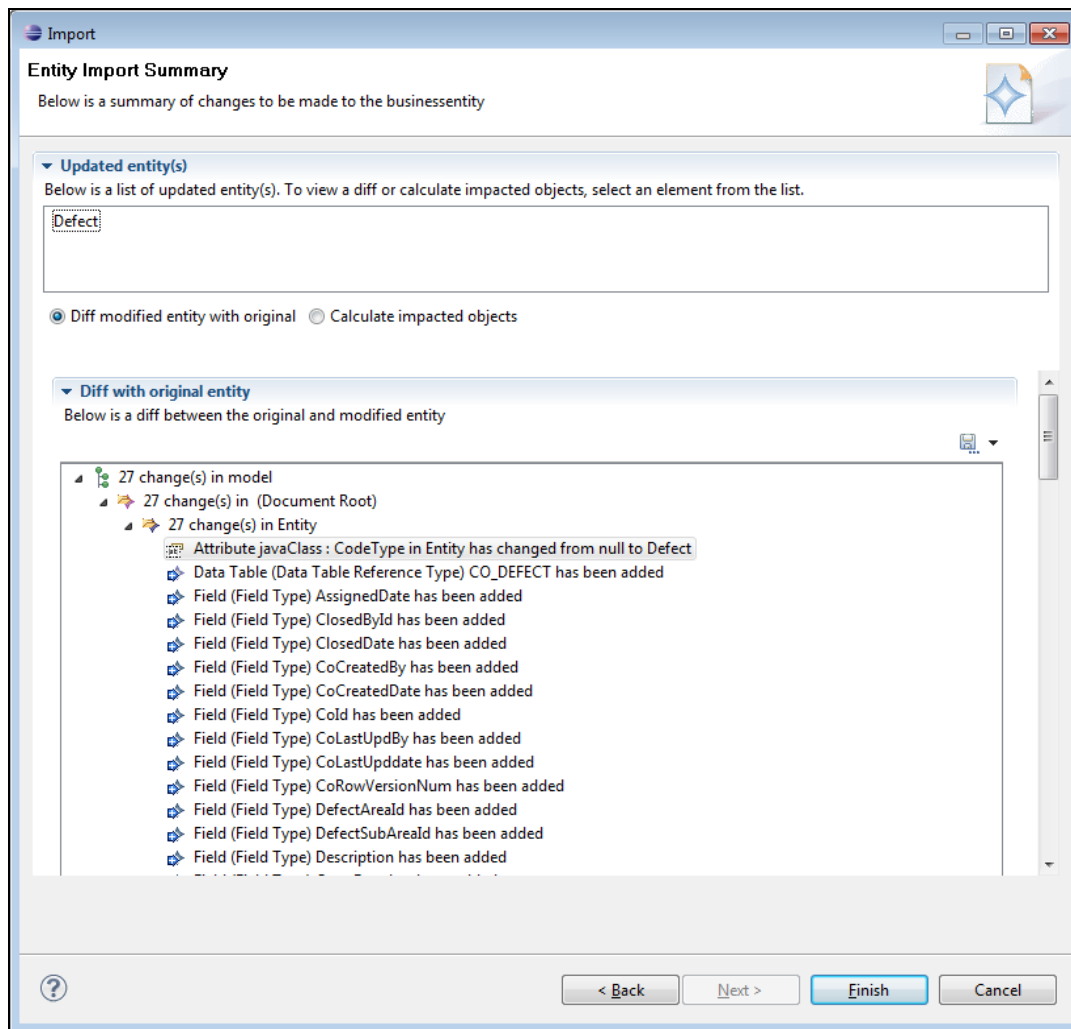
Entity	Data View	Data Table	Parent Entity	New
Defect (tutorial-model)		CO_DEFECT (tutorial-m...		☒

Create	Field Name	Column Name
☒	CoCreatedDate	CO_CREATED_DT
☒	Cold	CO_ID
☒	CoLastUpdBy	CO_LAST_UPD_BY
☒	CoLastUpddate	CO_LAST_UPDDT
☒	CoRowVersionNum	CO_ROW_VERSION_NUM
☒	DefectAreald	DEFECT_AREA_ID
☒	DefectSubAreald	DEFECT_SUB_AREA_ID
☒	Description	DESCRIPTION
☒	OpenDate	OPEN_DATE
☒	OwnedById	OWNED_BY_ID
☒	Priority	PRIORITY
☒	ReportedById	REPORTED_BY_ID
☒	Severity	SEVERITY
☒	Status	STATUS
☒	SubStatus	SUB_STATUS

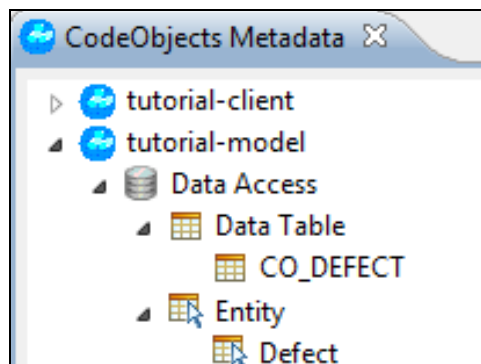
Fields Associations

< Back Next > Finish Cancel

6. Notice the many fields that are being added. Click **Finish**.



The new entity appears in the **CodeObjects Metadata** pane. If you see red X's, they will disappear after you add additional information and save the changes.



Define a Data Object for your Entity

The following are working definitions for the Data Object and Data Slice. We will explain in more detail later in the tutorial.

Data Object A metadata entry that is composed of one or more entities that form a logical data unit.

Data Slice The entity or entities within the data object that form a subset or slice of the original entity. For example, the Defect data object might include 10 fields, while a particular slice might include only 5 of those fields.

The data object will be transformed to a service that is accessible through SOA web service.

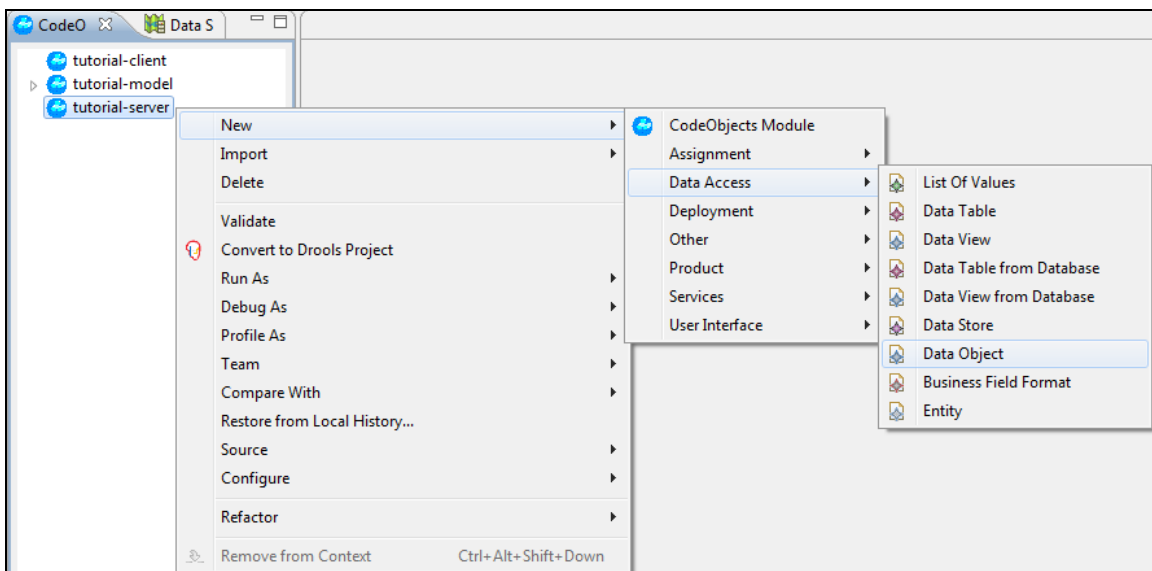
In this section, you will create a data object that consists of a simple Defect data slice. This data object will later be accessed through the web client to display a list of defects via the UI.

Note that you will use the **tutorial-server** module (not the **tutorial-model** module that you used to create the previous parts of the data layer) to contain your data object because although it is part of the data layer, it is used by the service layer to generate the service that will be deployed to the server side.

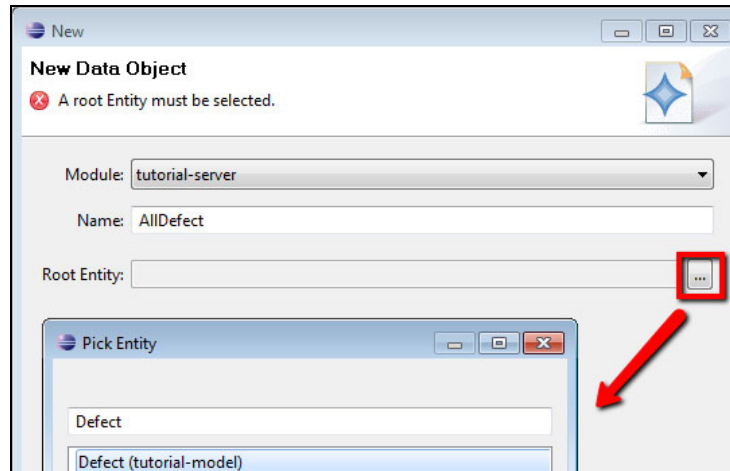
Create a Data Object

To create a data object:

1. In the Metadata pane, right-click the **tutorial-server** module and choose **New** → **Data Access** → **Data Object**.

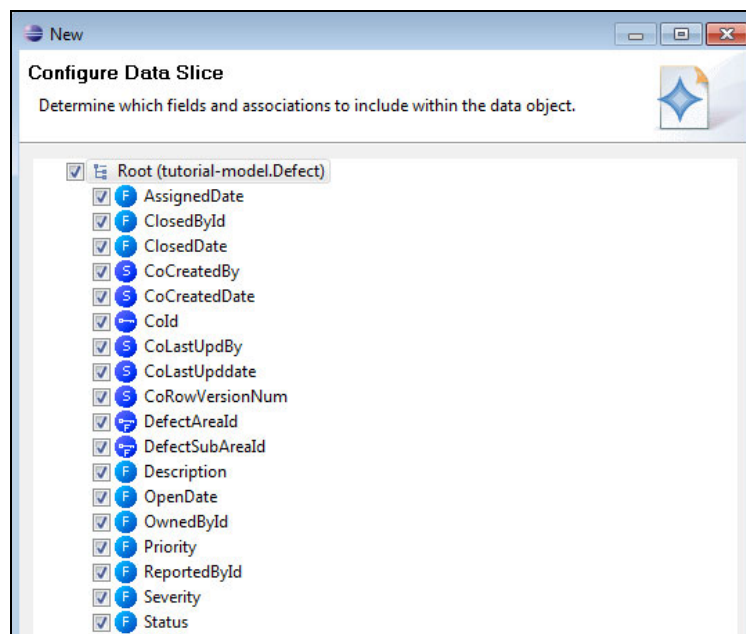


2. Enter the **Name** as **AllDefect** and click the **Pick** button [...] to set the **Root Entity**. In the **Pick Entity** box, start typing **Defect** to display only the entries that begin with the word *Defect*. Choose **Defect (tutorial-model)**, click **OK**, and click **Next**.



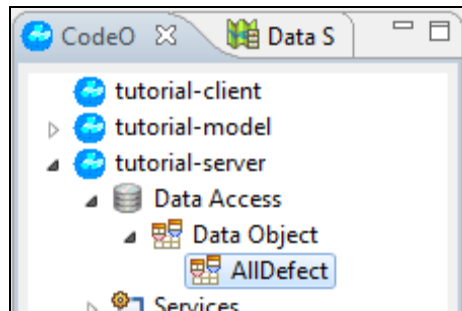
3. The next dialog box shows the fields available for UI display. This is where you define the shape or scope of data on which you intend to operate. To make all fields available for the defect list view, select all of the fields in the entity. Click the **Select All Fields** button. (If it is not available, click within the pane first.) Click **Finish**.

If you were to select only a limited number of fields, say 5 or 10, you would have a different data slice to use within the data object. The slice you take depends on the scope of data on which you plan to operate. For example, if you intend to update the field **Status**, you must include it.

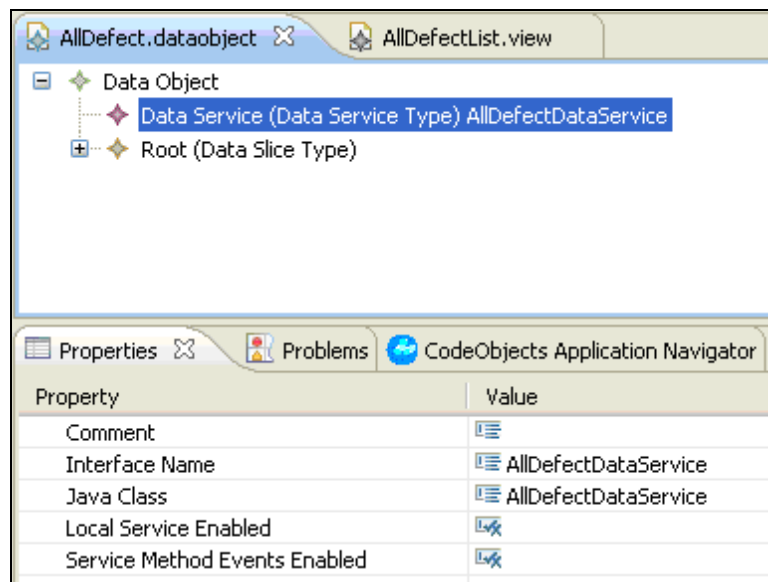


Defining the Data Layer

The newly created data object appears in the Metadata pane.



In the Metadata pane, double-click the **AllDefect** node. The Data Object appears in the Editor pane, and its properties appear below in the Property pane.



Defining the Services Layer

This chapter presents the services layer of the MDD model implemented by CodeObjects Platform.

High Level Metadata in the Services Layer

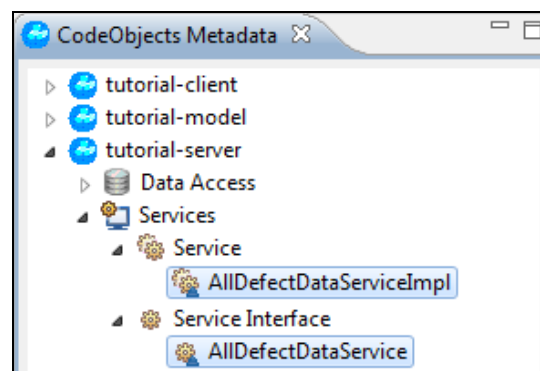
The high level metadata objects in the service layer appear in the **CodeObjects Metadata** pane. The high level metadata items are:

Service Interface Like the interface found in modern object-oriented programming languages, it defines the operations (or methods) with input and output parameters to which every implementation of the service interface should conform.

Service The service implementation of a service interface.

NOTE

For this tutorial, you use the default services generated in conjunction with the AllDefect data object, so you need not define or modify any service settings.



CORADS has generated one *service interface* and one *service*. When you created a data object in the previous step, that actually created these two service related metadata items behind the scenes.

Setting up for Server Deployment

With the service defined, in this chapter, you will set up the server side of the deployment layer.

You have created the data object with the generated services, and can proceed to deploy the service to your J2EE application server and see the live WSDL.

In this chapter you will:

1. Set up the standard folders and descriptor files that are required by J2EE application deployment.
2. Create a list of user roles for use in security settings in the System Model.
3. Create the **System** metadata and populate it with settings for the modules to be used, the services to include, and the security settings.
4. Create the **Application** metadata and populate it with settings for the type of deployment client, the deployment environment, and the application version.
5. Create a J2EE deployment file (WAR) and test it.

Metadata for Application and System

The high level metadata objects that you will create in this chapter will appear in the Metadata pane as you create them. They are:

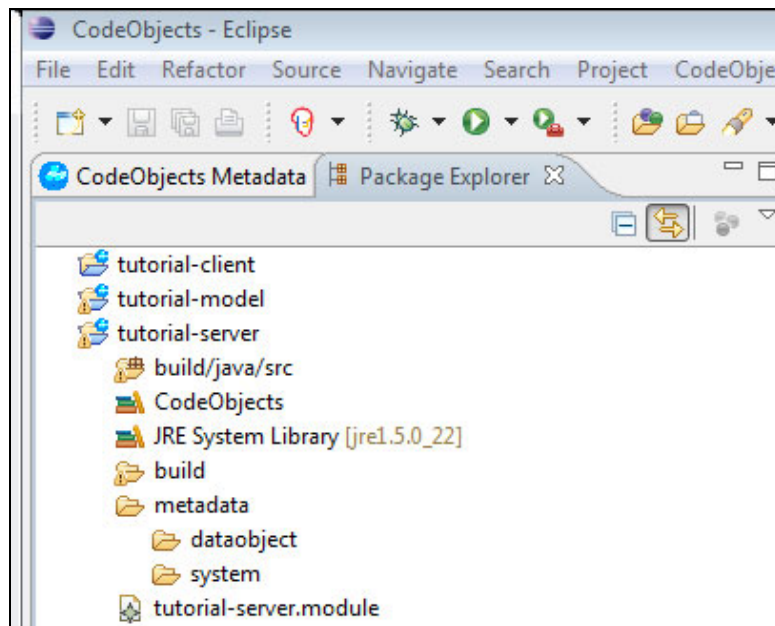
- **Application**
- **System**

Set up the Deployment Folder

The server side of CodeObjects Platform applications is deployed as a Web Application Archive (.WAR) file. The specification for WAR files is part of the JEE (formerly J2EE) standard. This is an archive or zip file that contains many files and folders. CORADS creates most of the content contained in the WAR file from your metadata. In this section, you will create folders and files for building a JEE application.

Switch to the Package Explorer

In the Metadata pane, click the **Package Explorer** tab. If it is not visible, click **Window** → **Show View** → **Package Explorer**.



The **Package Explorer** shows you the folder and file structure of your modules. (The **CodeObjects Metadata** view only showed you the metadata objects.) Folders correspond to the modules you created. Expand **tutorial-server** to see all the files under this folder. The **metadata** sub-folder contains the same metadata objects that you have been viewing in the **CodeObjects Metadata** view. CORADS generated all the other folders and files shown here, based on the metadata that you created.

TIP

The **Link With Editor** button  (an on/off toggle) near the top of the Package Explorer tab, links the Package Explorer with the Editor pane. Click a node in the Package Explorer to open the corresponding tab in the Editor pane. Likewise, click a tab in the Editor pane to highlight the corresponding node in the Package Explorer.

Setting up for Server Deployment

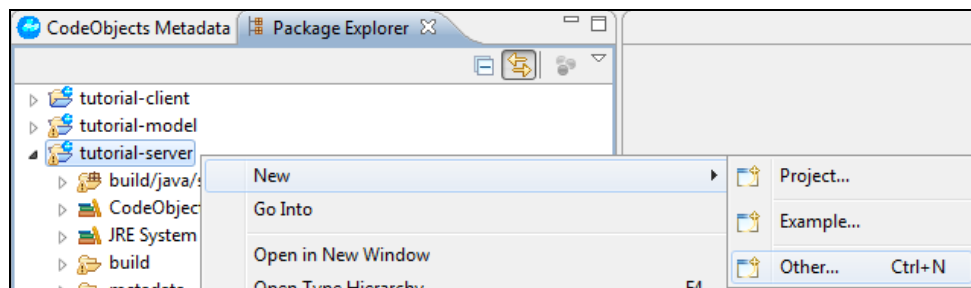
Create Deployment Folders

You must create two folders under **tutorial-server** in order to deploy a JEE application:

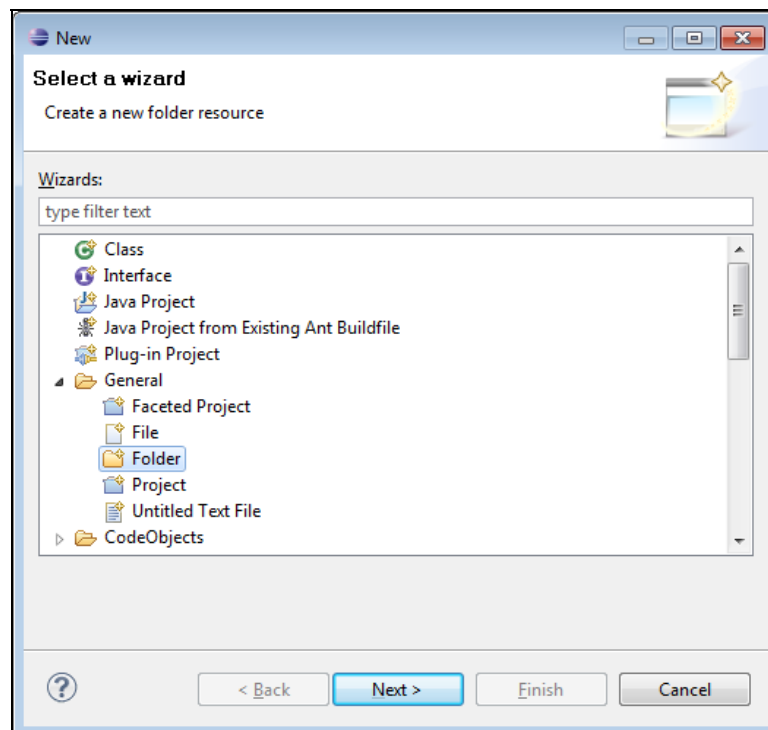
- **webapp** the deployment root
- **WEB-INF** required by the JEE standard

Create your deployment folder **webapp** under the **tutorial-server** folder.

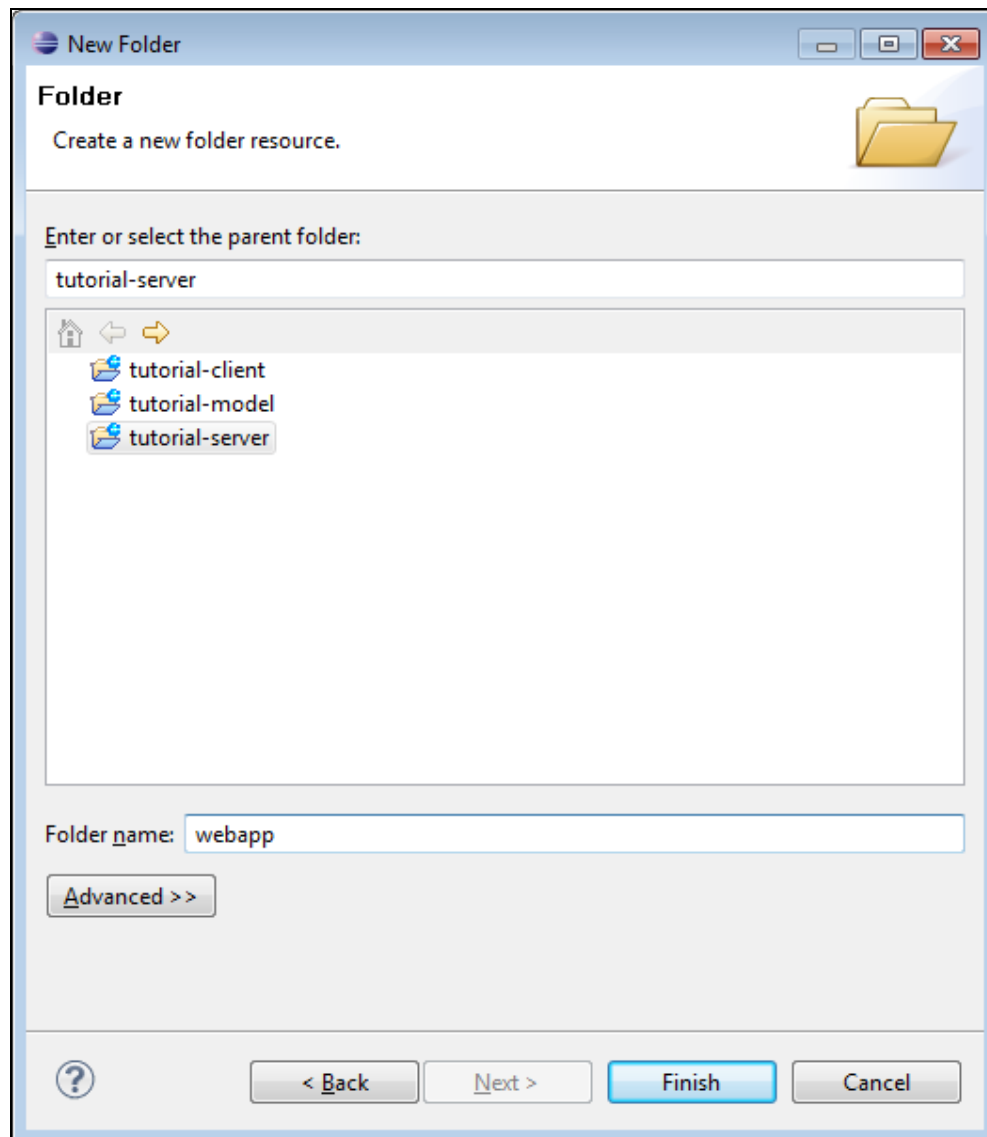
1. In the **Package Explorer**, right-click the **tutorial-server** folder. Recall that you placed the **tutorial-server** module in a folder called **tutorial-server** at the beginning of this tutorial.
2. Choose **New** → **Other**.



3. In the **New** dialog box, click **General** → **Folder**, and then click **Next**.



4. Be sure the parent folder is **tutorial-server**. If you opened this dialog box after selecting the **tutorial-server** folder in the **Package Explorer** view, this should be automatically set to that folder. Enter the **Folder name** as **webapp** and click **Finish**.



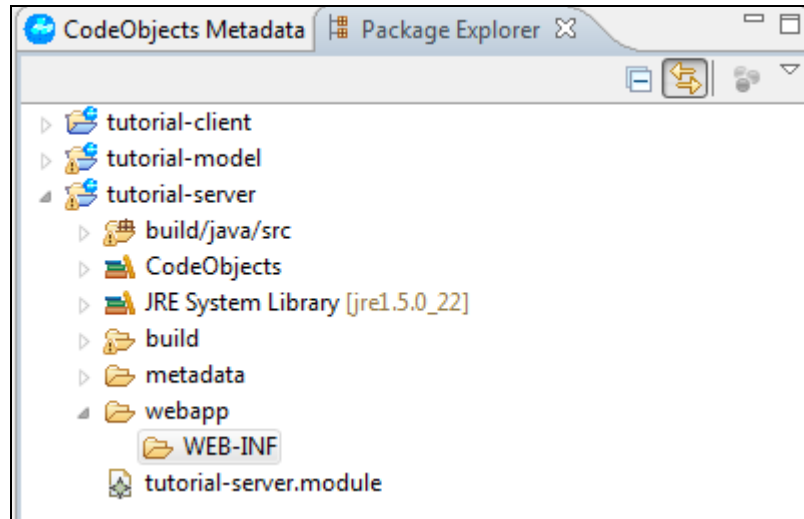
Create the WEB-INF Folder

You must create the WEB-INF folder for storing standard JEE components. Create this folder next as a subfolder of the **webapp** folder.

1. Right-click the **webapp** folder and choose **New** → **Other**.
2. Click **General** → **Folder** and click **Next**.

3. Enter the **Folder name** as **WEB-INF** and click **Finish** to create the folder under webapp.

The folder structure should look like this:

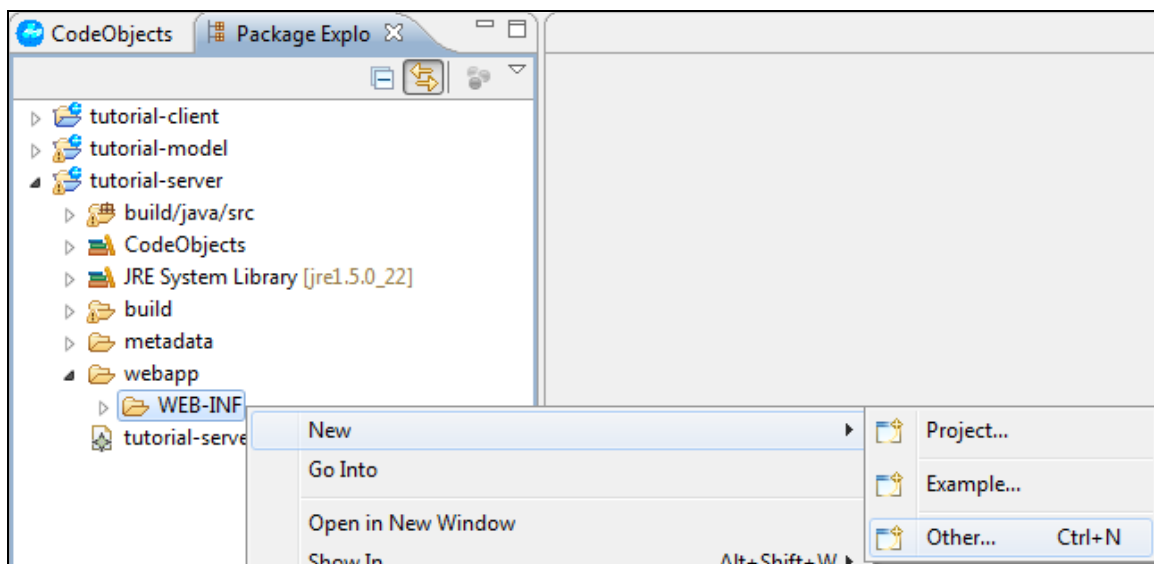


Create the Logging Configuration File

CodeObjects Platform standardizes all logging facilities through the open source logging framework [Apache Log4j](#). Log4j requires a configuration file **log4j.properties**.

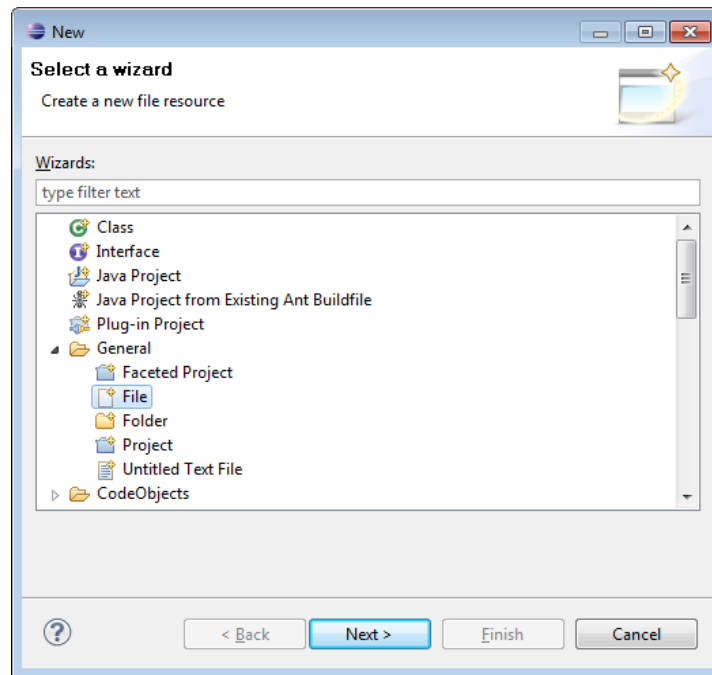
To create the file:

1. Right-click the **WEB-INF** folder and choose **New** → **Other**.

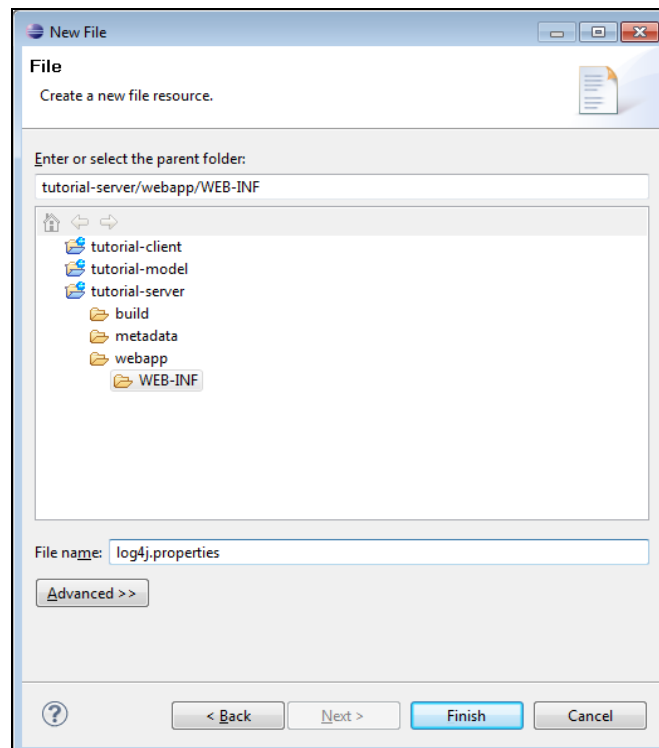


Setting up for Server Deployment

2. From the **New** dialog, click **General** → **File** and click **Next**.



3. Enter the **File name** as **log4j.properties**. Do not use a different file name. You will be referencing this file name later in the tutorial. Click **Finish** to create the file.



4. The working area of Eclipse now displays the editor and the file is open. Copy and paste the following text into the editor and click **File** → **Save** or press CTRL+S to save it.

```
log4j.rootLogger=WARN, stdout
log4j.logger.org.acegisecurity=WARN, codeobjects_security
log4j.logger.org.springframework=WARN, codeobjects_spring
log4j.logger.net.sf.hibernate=WARN, codeobjects_hibernate
log4j.logger.com.codeobjects=WARN, stdout
log4j.logger.com.codeobjects.services=INFO, codeobjects_services

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n

log4j.appender.codeobjects_spring=org.apache.log4j.DailyRollingFileAppender
log4j.appender.codeobjects_spring.File=${catalina.home}/logs/tutorial_spring.log
log4j.appender.codeobjects_spring.ImmediateFlush=true
log4j.appender.codeobjects_spring.DatePattern='.' yyyy-MM-dd
log4j.appender.codeobjects_spring.layout=org.apache.log4j.PatternLayout
log4j.appender.codeobjects_spring.layout.ConversionPattern=%d %p [%c] - %m%n

log4j.appender.codeobjects_services=org.apache.log4j.DailyRollingFileAppender
log4j.appender.codeobjects_services.File=${catalina.home}/logs/tutorial_services_log
log4j.appender.codeobjects_services.ImmediateFlush=true
log4j.appender.codeobjects_services.DatePattern='.' yyyy-MM-dd
log4j.appender.codeobjects_services.layout=org.apache.log4j.PatternLayout
log4j.appender.codeobjects_services.layout.ConversionPattern=%d\t\t\t\t%X{UserId}\t\tX{SessionId}\t%p\t%c\t%m%n

log4j.appender.codeobjects_security=org.apache.log4j.DailyRollingFileAppender
log4j.appender.codeobjects_security.File=${catalina.home}/logs/tutorial_security_log
log4j.appender.codeobjects_security.ImmediateFlush=true
log4j.appender.codeobjects_security.DatePattern='.' yyyy-MM-dd
log4j.appender.codeobjects_security.layout=org.apache.log4j.PatternLayout
log4j.appender.codeobjects_security.layout.ConversionPattern=%d %p [%c] - %m%n

log4j.appender.codeobjects_hibernate=org.apache.log4j.DailyRollingFileAppender
log4j.appender.codeobjects_hibernate.File=${catalina.home}/logs/cohomewarranty_hibernate.log
log4j.appender.codeobjects_hibernate.ImmediateFlush=true
log4j.appender.codeobjects_hibernate.DatePattern='.' yyyy-MM-dd
log4j.appender.codeobjects_hibernate.layout=org.apache.log4j.PatternLayout
log4j.appender.codeobjects_hibernate.layout.ConversionPattern=%d %p [%c] - %m%n
```

You have completed the server-side J2EE deployment setup. Before going to the next section, be sure that this file appears under the `webapp/WEB-INF` folder by checking for it in the Metadata pane under the Package Explorer tab.

Set up Security

You will create two users—*couser* for a regular user and *coadmin* for an administrator. You will also define two user roles: *user* and *administrator* for use later in the tutorial.

Create the User Credentials File

For this tutorial application, you will use a simple memory-based user authentication for demonstration purposes. To do that, you will store the user names and passwords in a file called *user.properties* in the **WEB-INF** folder.

NOTE

CodeObjects Platform provides several types of authentication mechanism. This in-memory authentication model is just one of them, used in the tutorial.

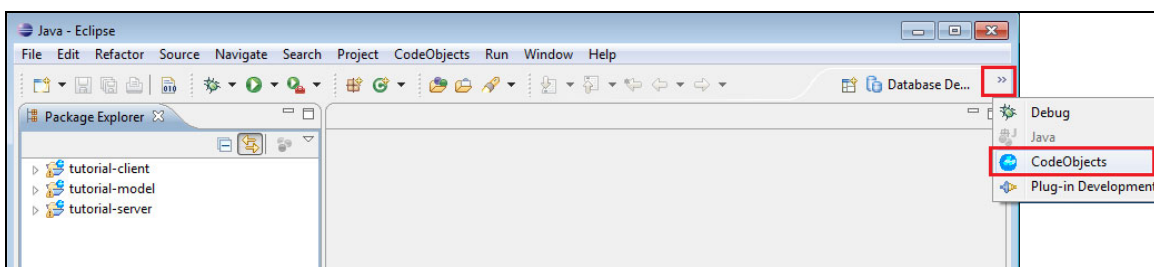
To create **WEB-INF/user.properties**

1. Right-click the **WEB-INF** folder and choose **New** → **Other**.
2. In the **New** dialog, click **General** → **File** and click **Next**.
3. Enter the file name as **user.properties**. Do not use a different file name. You will refer to it elsewhere in this tutorial. Click **Finish** to create the file.
4. Copy and paste the following text into the editor and click **File** → **Save** or press CTRL+S to save it. The content below will allow for logins with the two names *coadmin* and *couser*. The password for each is the same as the login names. Although you can change these values at any time, do not do so now because you will use these users later in the tutorial.

```
coadmin=coadmin,_ROLE_AUTHENTICATED_USER
couser=couser,_ROLE_AUTHENTICATED_USER
```

Create User Roles

Switch back to the CodeObjects Perspective layout. To do this quickly, at the upper right of the toolbar, click the Right Arrows >> and choose **CodeObjects** (if it is available.)



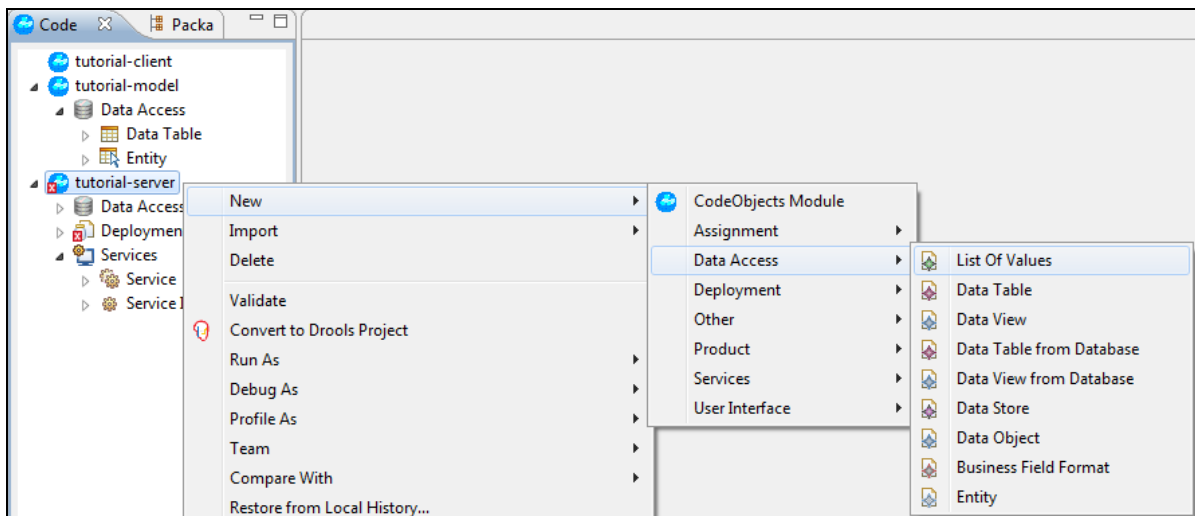
Setting up for Server Deployment

You will define two user access roles that will be used later in other tutorial sections, but here you are just setting them up:

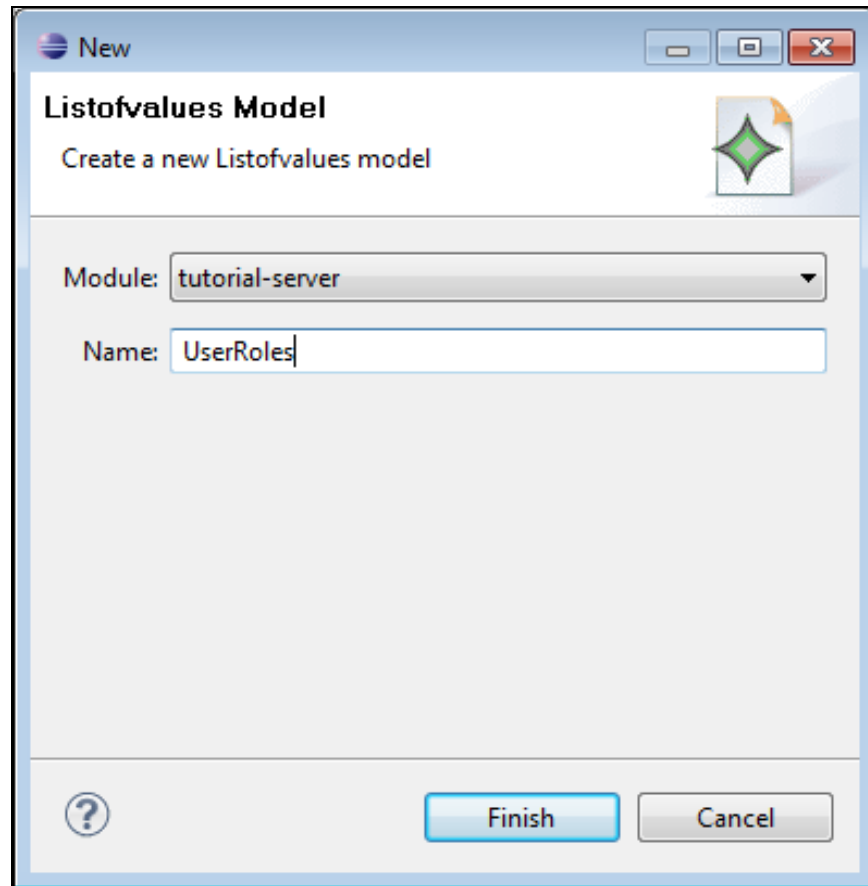
- User
- Administrator

To define the roles, create a list:

1. In the Metadata pane, click the **CodeObjects Metadata** tab.
2. Right-click the **tutorial-server** module and choose **New** → **Data Access** → **List Of Values**.

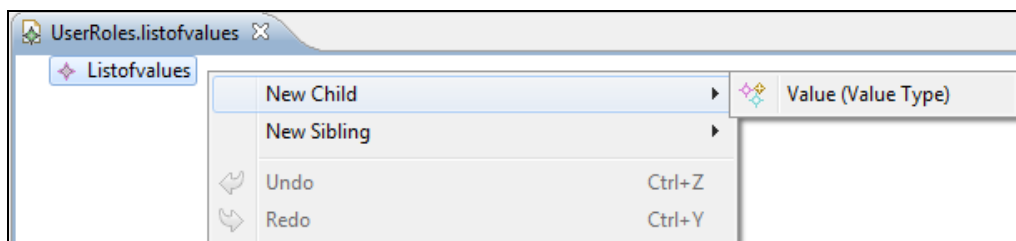


3. For **Name**, enter **UserRoles**, and then click **Finish**.

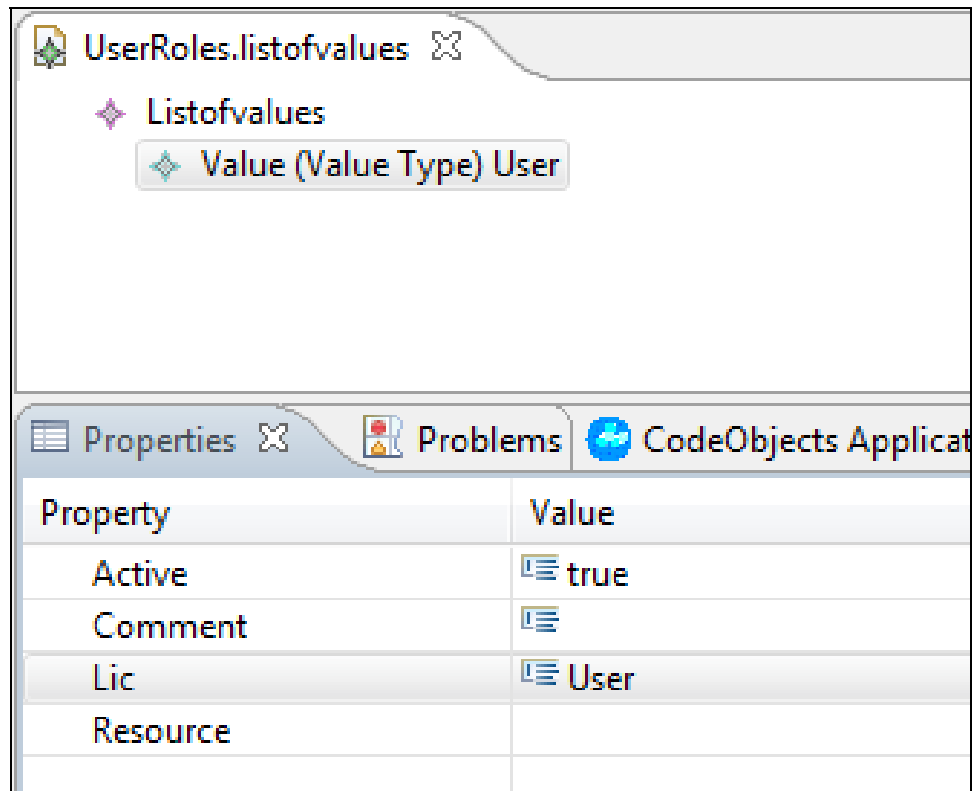


The list is created in the Metadata pane under **tutorial-server** → **Data Access**, and a list of values node appears in the Editor pane. To add an entry in the list for the User role:

1. In the Editor pane, right-click the **Listofvalues** node and choose **New Child** → **Value (Value Type)**.

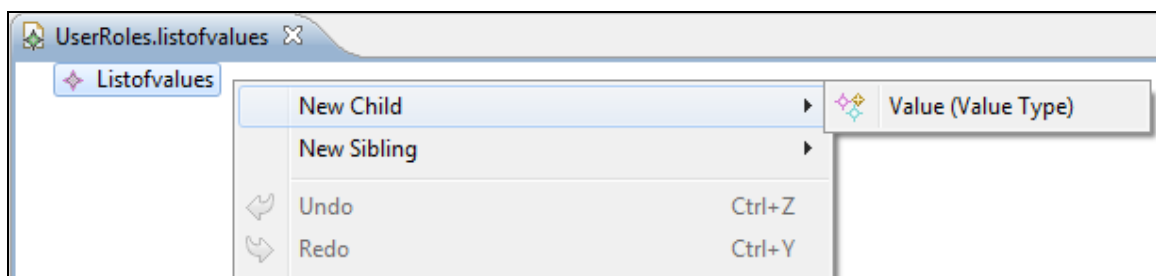


2. In the Property pane, under the **Properties** tab, for the **Property** column of **Lic** (Language Independent Code), enter its **Value** as **User**.



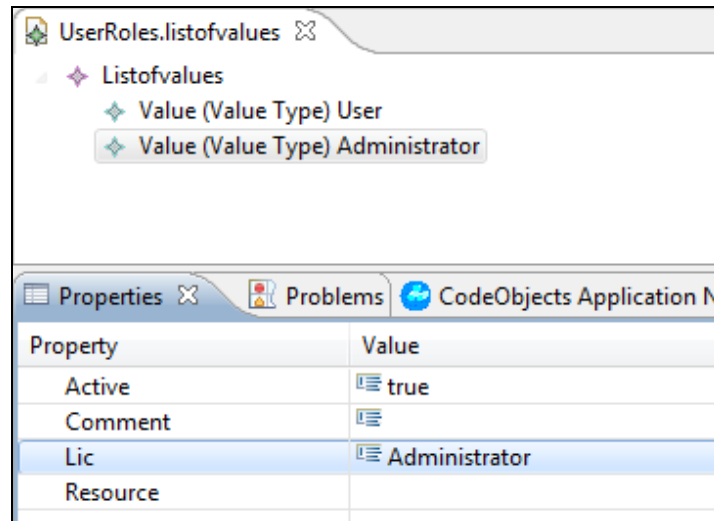
Add an entry in the list for the Administrator role:

3. In the Editor pane, right-click the **Listofvalues** node (not the **Value...** node) and choose **New Child** → **Value (Value Type)**.



4. In the Property pane, for the **Property** of **Lic** (Language Independent Code), set the **Value** to **Administrator**.

The **Lic** field is required for all lists of values.

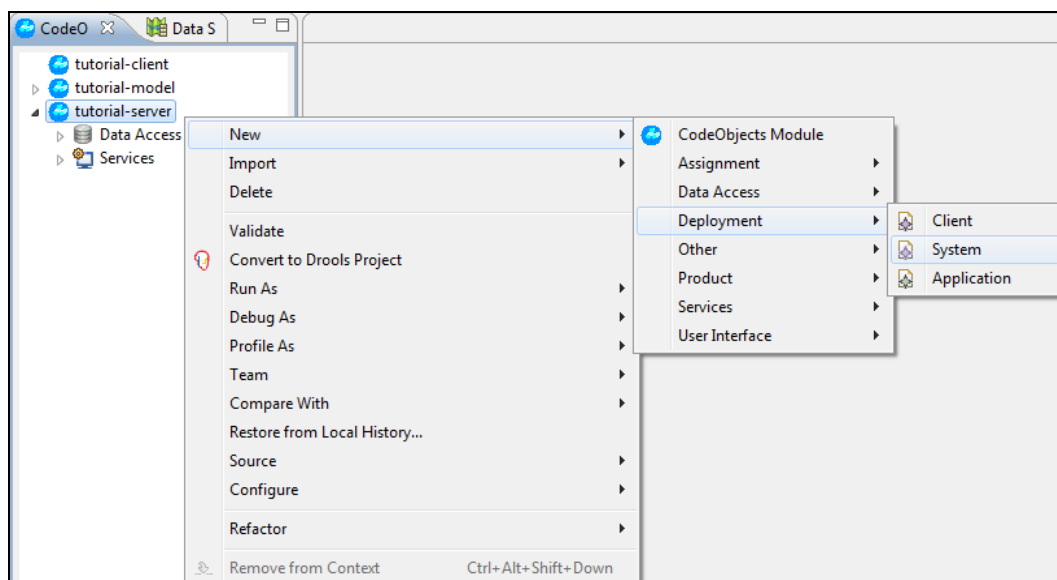


Save your changes. Click **File** → **Save** or press CTRL+S.

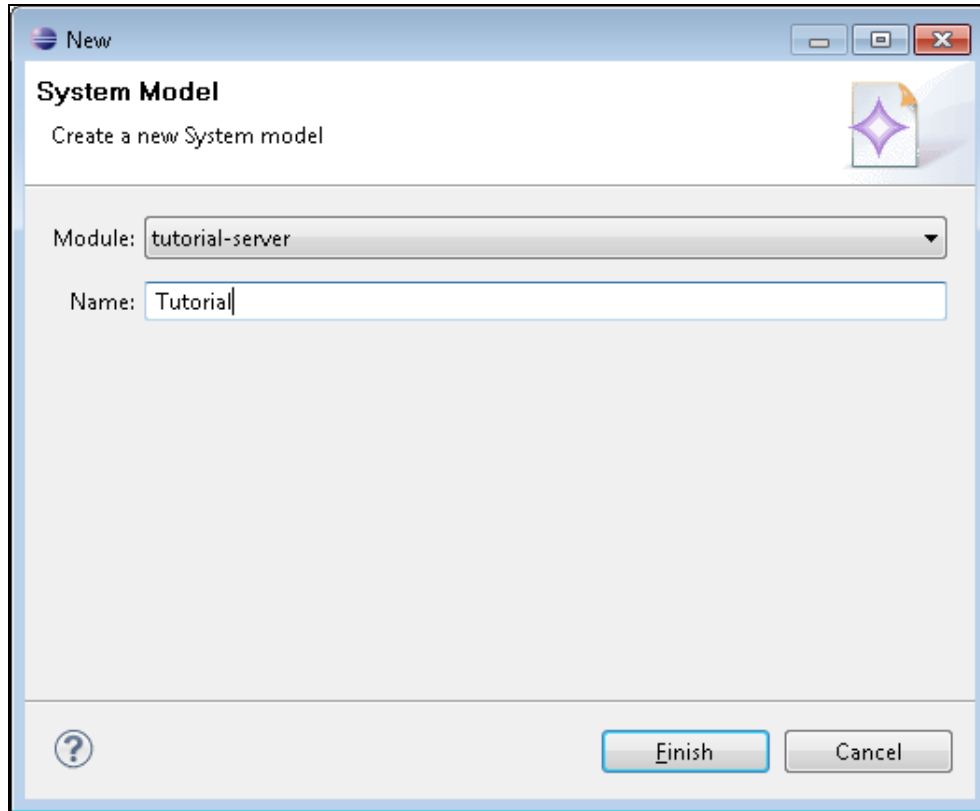
Create the System Metadata

The **System** metadata is used for settings for the modules to be used, the services to include, and the security settings. To create the **System** metadata entry:

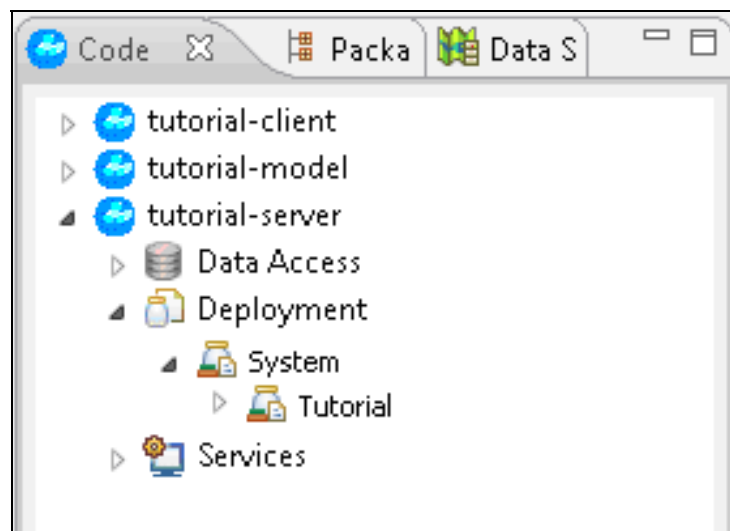
1. In the Metadata pane, right-click **tutorial-server** and choose **New** → **Deployment** → **System**.



2. In the **Name** field, enter **Tutorial** and click **Finish**. This creates the *Tutorial.system* metadata entry that you will use to set system parameters.



In the Metadata pane under **tutorial-server**, a new node appears:
Deployment → **System** → **Tutorial**.

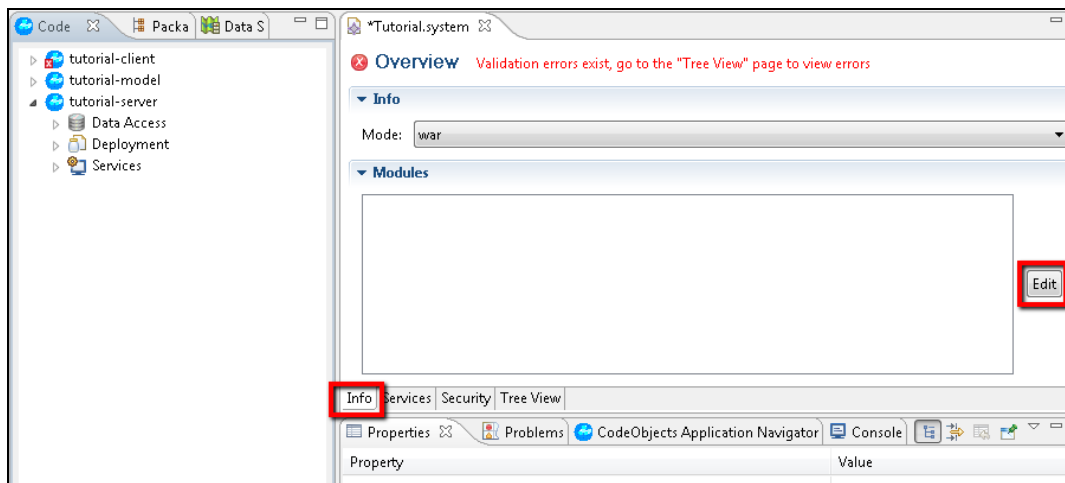


Setting up for Server Deployment

Add Modules to the System

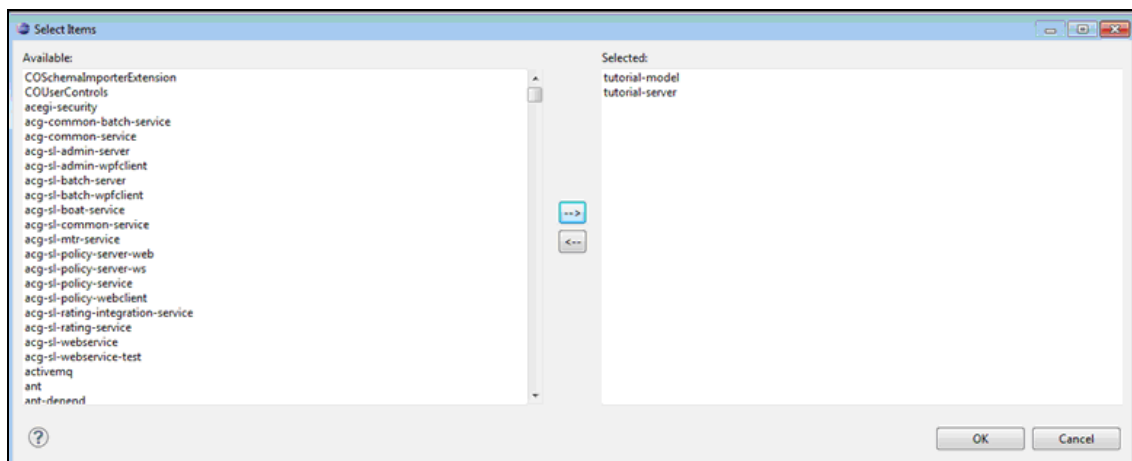
In the Editor pane, **Tutorial.system** has opened. To specify which modules will be included in your system:

1. At the bottom of the Editor pane, click the **Info** tab, and then click the **Edit** button.



2. Select two tutorial modules:
 - a. Click the module **tutorial-model**
 - b. Press and hold CTRL and click **tutorial-server** (to select both of them.)
 - c. Release CTRL and click the **Right Arrow**  to move them from the **Available** pane to the **Selected** pane.
 - d. Click **OK**.

This sets the scope for the CORADS system metadata, which will primarily be based on these two modules. Therefore CORADS will focus on these two modules when editing system metadata.



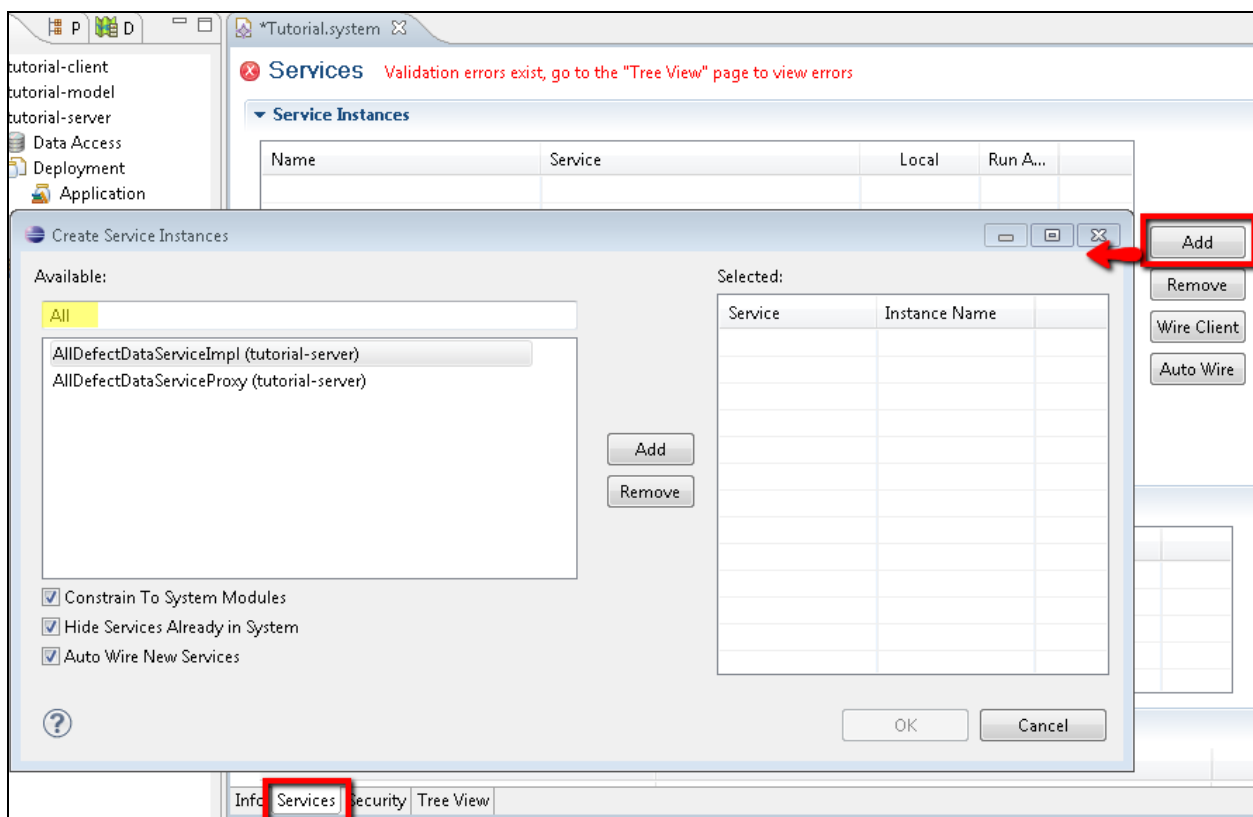
Add Services to the System Model

Recall that when you set up the business logic (services) layer metadata, you created a data object called **AllDefect**. You are going to add that data object/service to your system and add an authentication service as well.

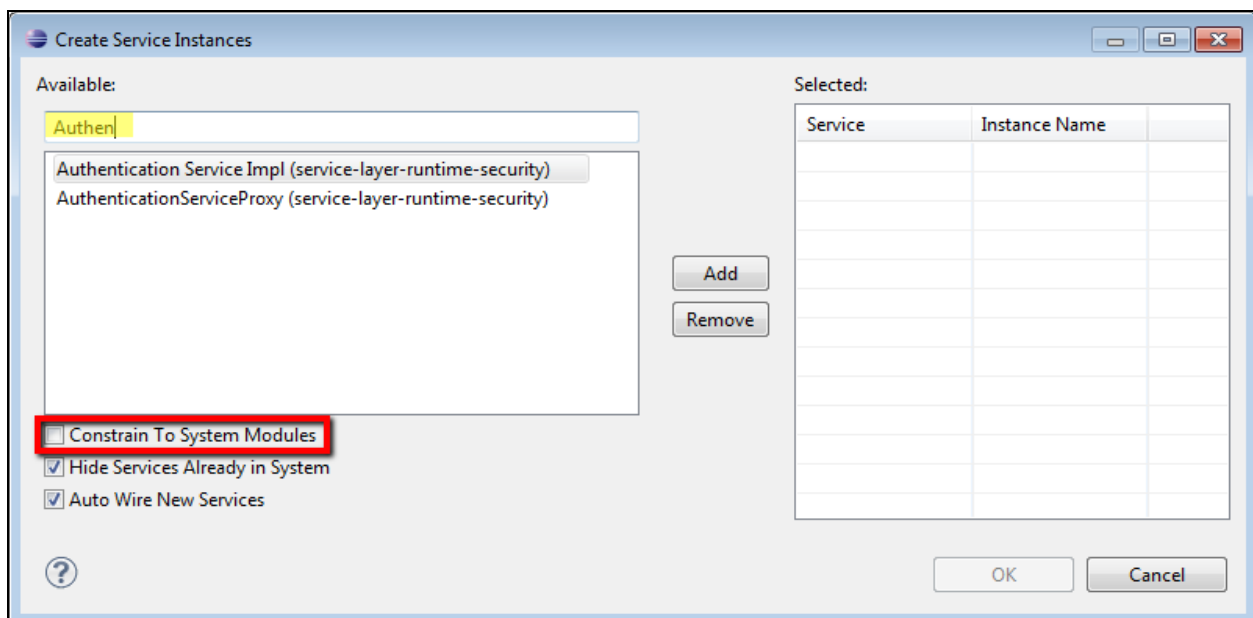
To add those services to your system model:

1. Click the **Services** tab, and then click the **Add** button.
2. Select **AllDefectDataServiceImpl (tutorial-server)** and click **Add**. Click **OK** to add the service.

Because you entered the two modules, tutorial-model and tutorial-server, CORADS narrows the search to focus on these two modules to find the matching service names. If you do *not* include these two modules, you can clear the **Constrain to System Modules** check box to broaden the search scope beyond the modules you entered at the earlier step.

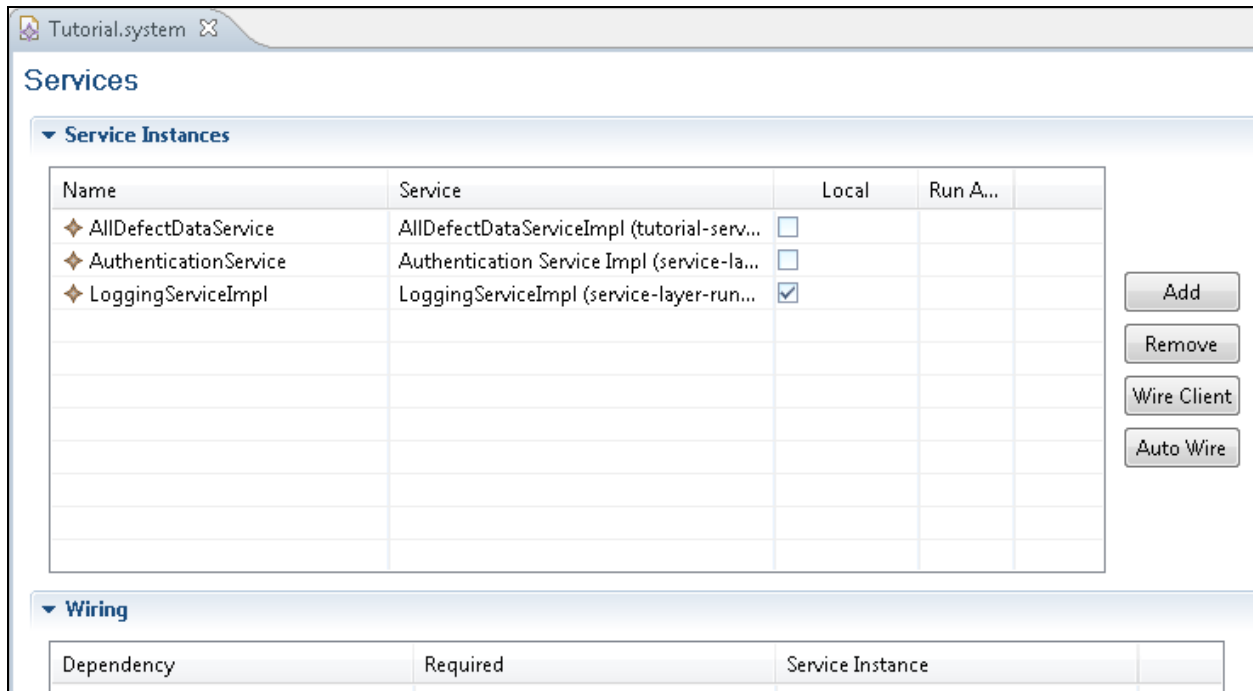


3. Still on the **Services** tab, click **Add**.
 - a. Clear the **Constrain to System Modules** check box.
 - b. In the text box under **Available**, type **Authentication** to filter the module list. Note that the letters you type filter the **Available** list.
 - c. Click **Authentication Service Impl (service-layer-runtime-security)**, and then click **Add**, to move it to the Selected pane.
 - d. Click **OK**.

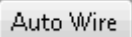


This includes the pre-created authentication service to the system. CodeObjects has pre-built many useful services for you to include in your system metadata. This Authentication Service Impl service is an implementation of web application authentication.

The **Service Instances** box displays three service instances.



NOTE

Wiring is the CORADS process for setting dependencies for a service. For example, **AllDefectDataService** depends on the **Logging Service** to write logs. Such a dependency must be specified in order for the logging feature to work correctly. Whenever you select and add a service to the system metadata, CORADS performs *Auto Wiring* to set all the dependencies for you. Optionally, you can click the **Auto Wire** button  to do the same thing.

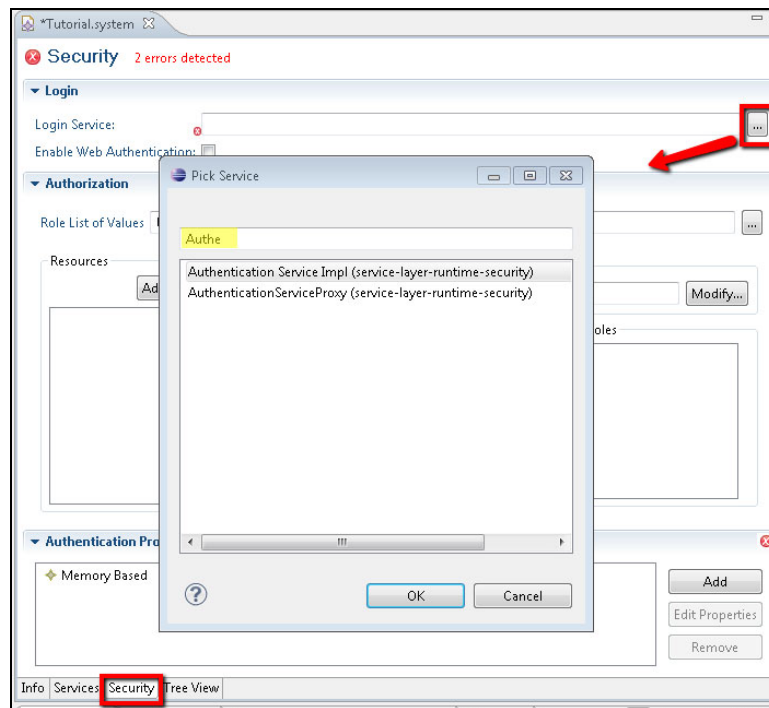
Set up Security for the System Model

The last step in setting up your System metadata is to specify the security settings.

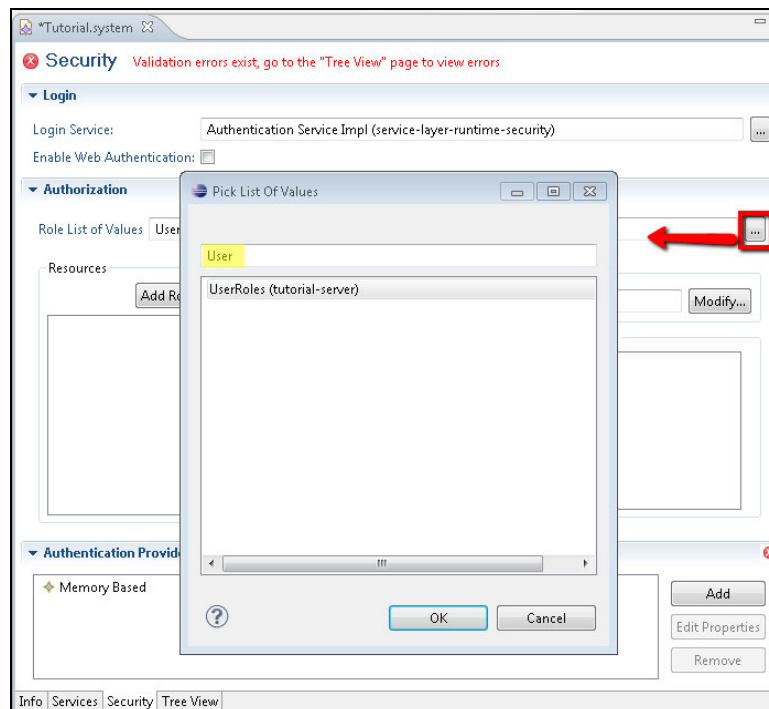
1. At the bottom of the Editor pane, click the **Security** tab, and on the right of the **Login Service** field, click the **Pick** button.
2. From the **Pick Service** dialog, type the partial keyword **Authe** (note that the list filters as you type each letter) and click **Authentication Service Impl (service-layer-runtime-security)**.

Setting up for Server Deployment

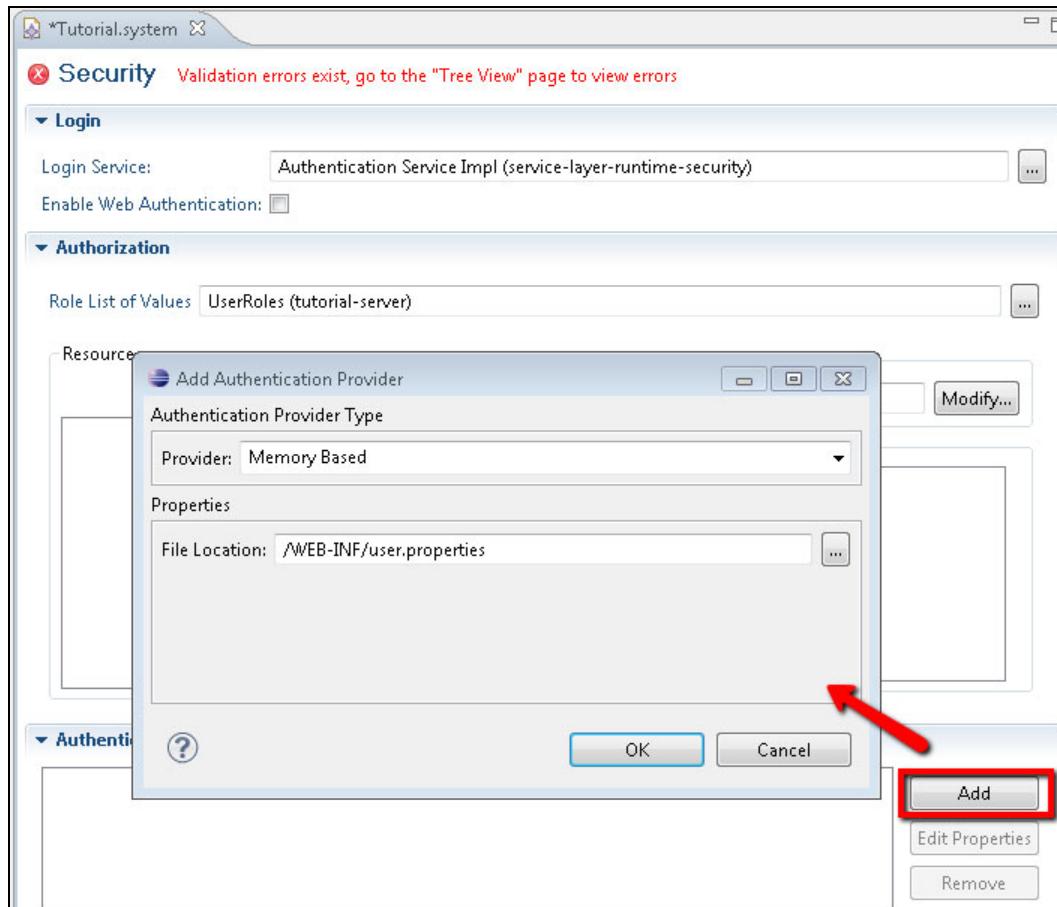
3. Click **OK**.



4. On the right of **Role List of Values**, click the **Pick** button.
From the list choose **User Roles (tutorial-server)**, and then click **OK**.



- Click the **Add** button to open the **Add Authentication Provider** dialog box. For **Provider** click the **List** arrow and choose **Memory Based**, and for the **File Location**, click the **Pick** button to choose **WEB-INF/user.properties**, which is the file that you created in a previous step in this tutorial. Click **OK**.

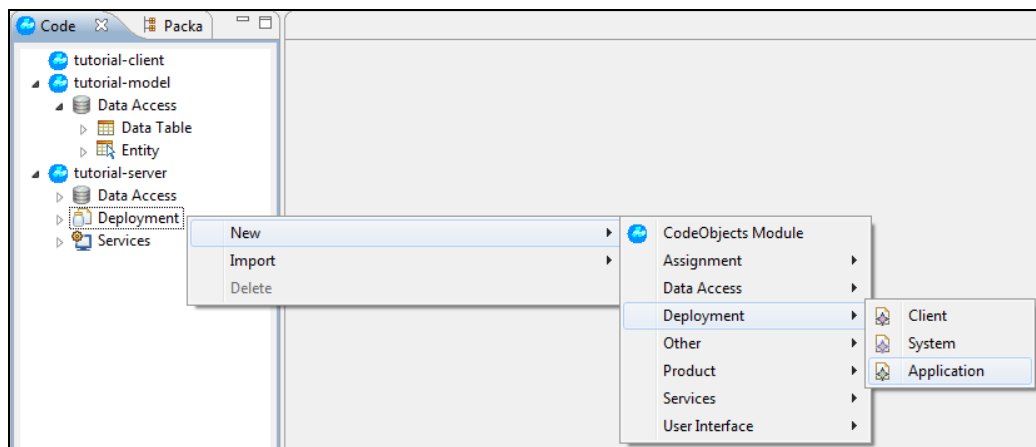


- Save your settings. Click **File** → **Save** or press CTRL+S.

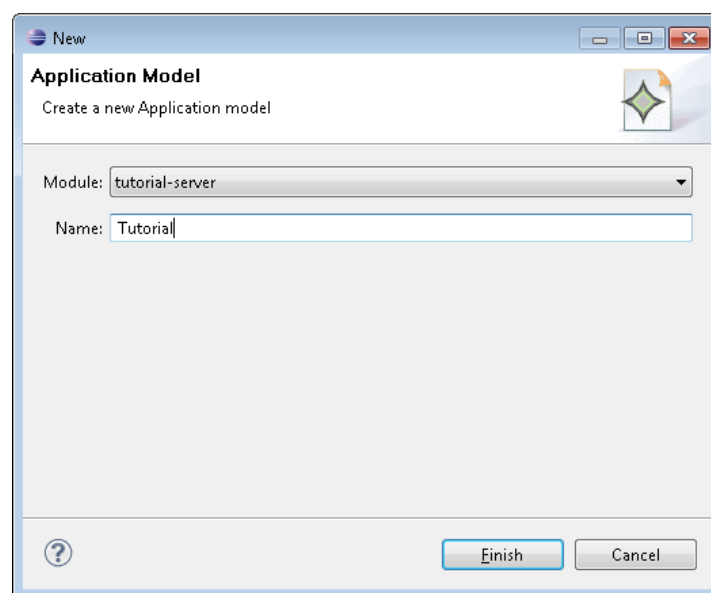
Create the Application Metadata

The last major portion of metadata to specify for the application is the **Application** metadata node, which enables us to specify settings for the type of deployment client, the deployment environment, and the application version. To create and populate the **Application** metadata:

1. In the Metadata pane, right-click **tutorial-server** and choose **New** → **Deployment** → **Application**.



2. In the **Name** field, enter **Tutorial** and click **Finish**.



The initial application is created and appears in the Metadata pane under **tutorial-server** → **Deployment** → **Application**.

Set Properties of the Application

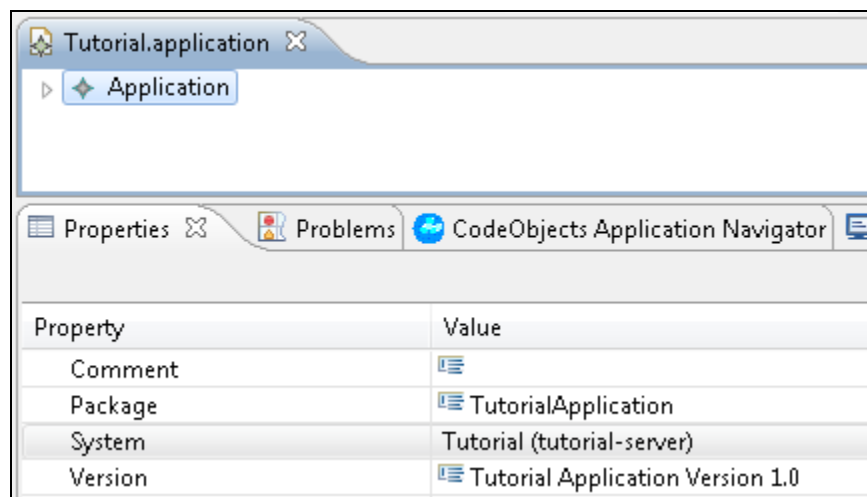
In the Editor pane, if the **Tutorial.application** tab is not already the active tab, click it. To set the parameters:

1. In the Editor pane, click the **Application** node to display the application properties in the Property pane. If the **Properties** tab is not already the active tab, click it. Set the following:

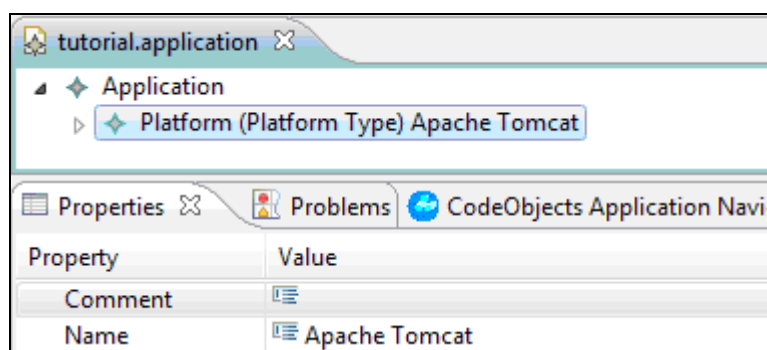
Package **TutorialApplication**

System Choose the Tutorial system created in the previous step. Double-click the Property or its Value field to access the **Pick Property** button and from the dialog box, pick **Tutorial (tutorial-server)**.

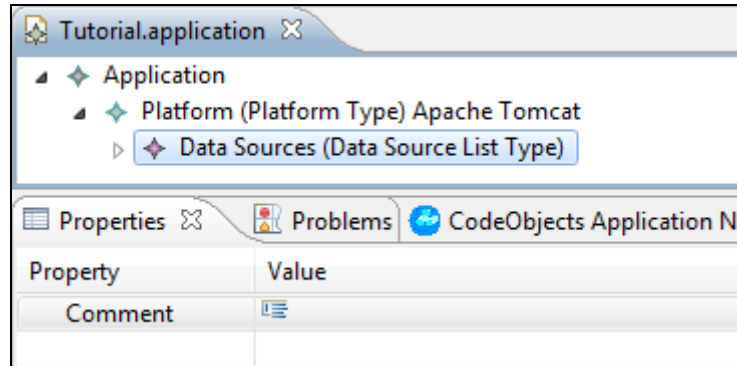
Version This can be any value meaningful to you per your versioning scheme. In the image, it is set to **Tutorial Application Version 1.0**



2. In the Editor pane, right-click **Application** and choose **New Child → Platform (Platform Type)**. In the Property pane, set the property **Name** to **Apache Tomcat**. (Tomcat is our test application server.)



3. In the Editor pane, right-click the **Platform** node and choose **New Child** → **Data Sources (Data Source List Type)**.



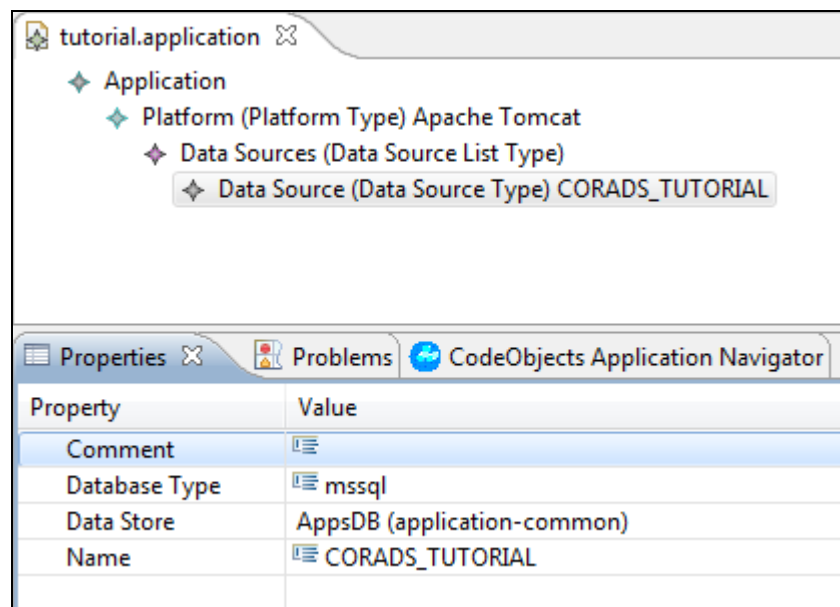
4. Right-click **Data Sources...** and choose **New Child** → **Data Source (Data Source Type)**.
In the Property pane, set the properties:

Database Type **mssql**

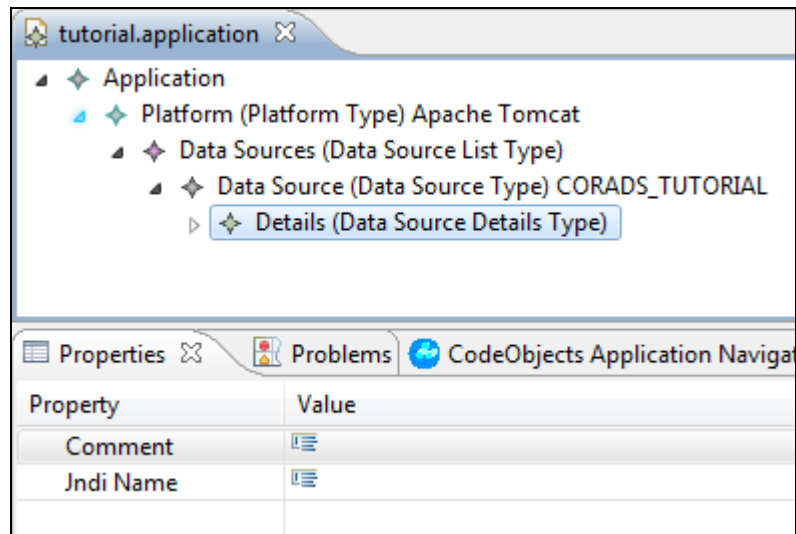
Data Store **AppsDB (application-common)**

Name **CORADS_TUTORIAL**

Note that the **Name** can be anything, but we recommend that you make it the same as the database to which you are connecting.



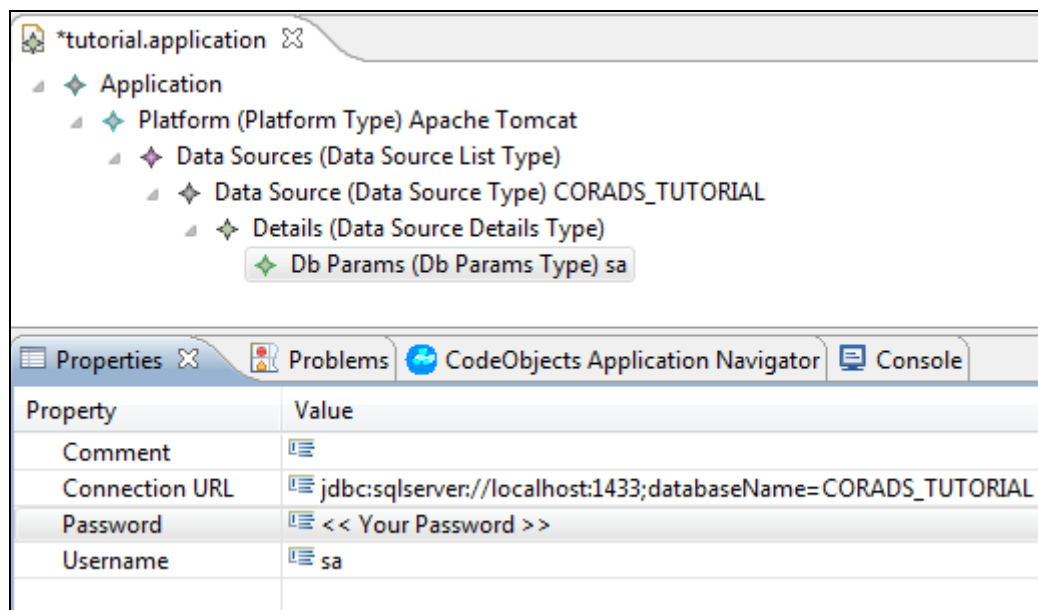
- Right-click **Data Source...** and choose **New Child → Details (Data Source Details Type)**.



- Right-click **Details...** and choose **New Child → Db Params (Db Params Type)**. Set the properties to:

Connection URL **jdbc:sqlserver://server;databaseName=CORADS_TUTORIAL**
(replace *server* with your server name)

Username/Password (The username and password for your database.)



- Save your changes. Click **File → Save** or press CTRL+S.

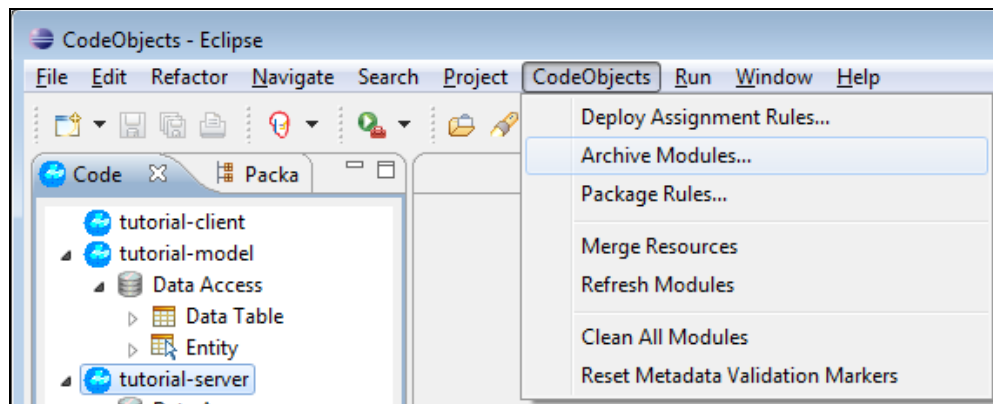
Deploying the Server

In this chapter, you will generate a .WAR file that will contain the server module so you can deploy and test your service.

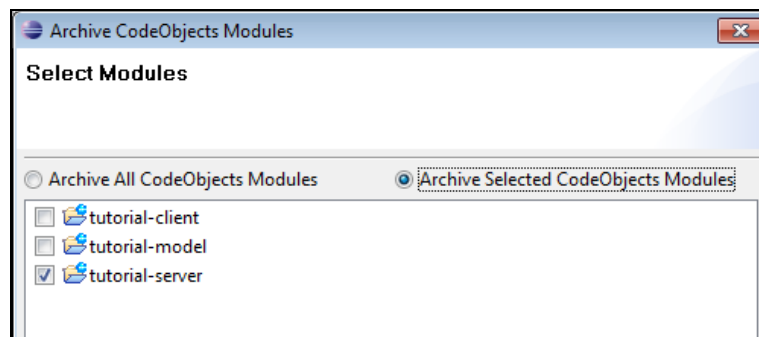
Create a Deployable Web Archive File

Create the deployable WAR (Web Archive) file.

1. From the menu, click **CodeObjects** → **Archive Modules**.

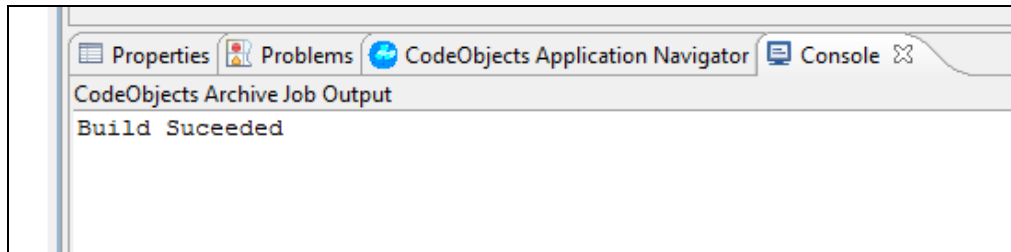


2. Select the **tutorial-server** module check box and click **OK**.



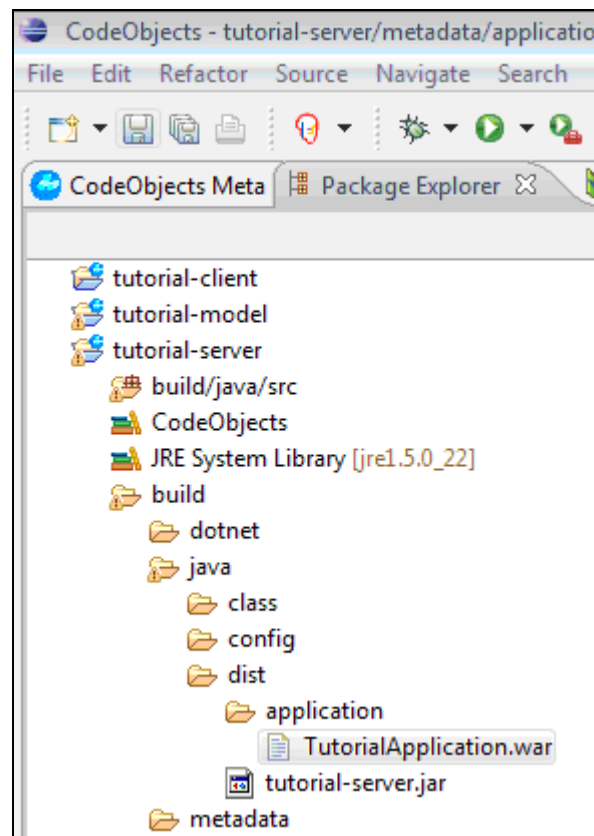
Deploying the Server

When the file is created, in the Property pane under the **Console** tab, the message should indicate **Build Succeeded**.



If the build fails, the Console will display the error, which you can use to troubleshoot and repair any metadata errors.

3. In the Metadata pane, click the **Package Explorer** tab and navigate down through the folder **tutorial-server** → **build** → **java** → **dist** → **application**, where CORADS has created the **TutorialApplication.war** file. This is the WAR file that you can deploy to the Tomcat application server.



Test the Deployment Files

Let's test the WAR file that was generated. You must use Tomcat to test, and it's beyond the scope of this document to show you how to install and use Tomcat. Refer to the Tomcat documentation for details. Below is the high level user description.

Copy the File

Copy **TutorialApplication.war** into your Tomcat installation folder's **webapps** folder.

If the Tomcat server is already running, and you have deployed a WAR file before, be sure to stop Tomcat and in the webapps folder, delete both the old deployment WAR file and the Tutorial Application folder before copying the new file and restarting Tomcat.

TIP

With the Package Explorer already open and expanded to show the file, simply right-click the *.war* file and choose **Copy**. Open Windows Explorer, navigate to the *webapps* folder, and right-click and choose **Paste** or press CTRL+V. You can overwrite the previous version, if any. Just remember to delete any previous application folder (for the tutorials, it is named either *TutorialApplication* or *TutorialApplicationWebClient*.)

The next two steps will help verify that the deployment was successful.

Verify the AllDefect Data Object's WSDL

If you are running Tomcat on your computer, you can access this URL:

<http://localhost:8080/TutorialApplication/service/AllDefectDataService?wsdl>

to see the WSDL for the **AllDefect** service. Adjust the URL appropriately if you are running the test server on another computer.

Verify the Authentication Service's WSDL

Similarly, you can access this URL:

<http://localhost:8080/TutorialApplication/service/AuthenticationService?wsdl>

(or the URL for your test computer) to see the WSDL for the authentication service.

Congratulations! You have successfully built the first WAR file using CORADS. You will now proceed to build the user interface layer.

Defining the User Interface Layer

Having configured and built the server side of your application, you must define the client interface.

In this chapter, you will:

1. Create a view to display the defects data.
2. Build a widget. Widgets are containers for collections of views.

Every client requires at least one widget and at least one view. Widgets correspond to tabs on the screen, and views present data to your end users through standard UI controls.

High Level Metadata in the Client/UI Layer

The high level metadata objects that you create in this chapter will appear in the **CodeObjects Metadata** pane as you create them. They are:

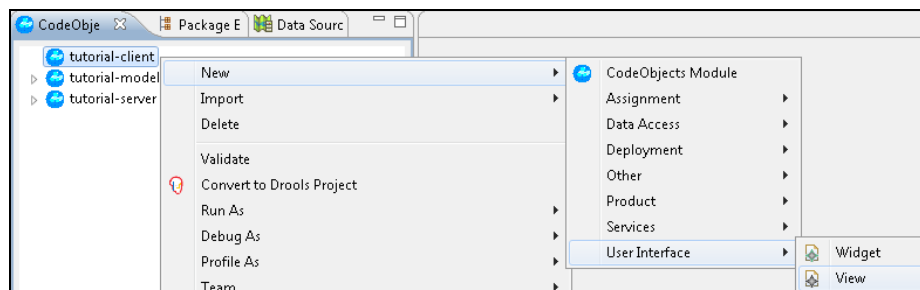
- **Widget**
- **View**

Create a View

In this section, you will create a simple list view to show a list of defects, using the **tutorial-client** module. Recall that this is the client-side module that encapsulates all user-interface-related and client-side-related metadata.

Create the new **View** metadata, **AllDefectList.view**:

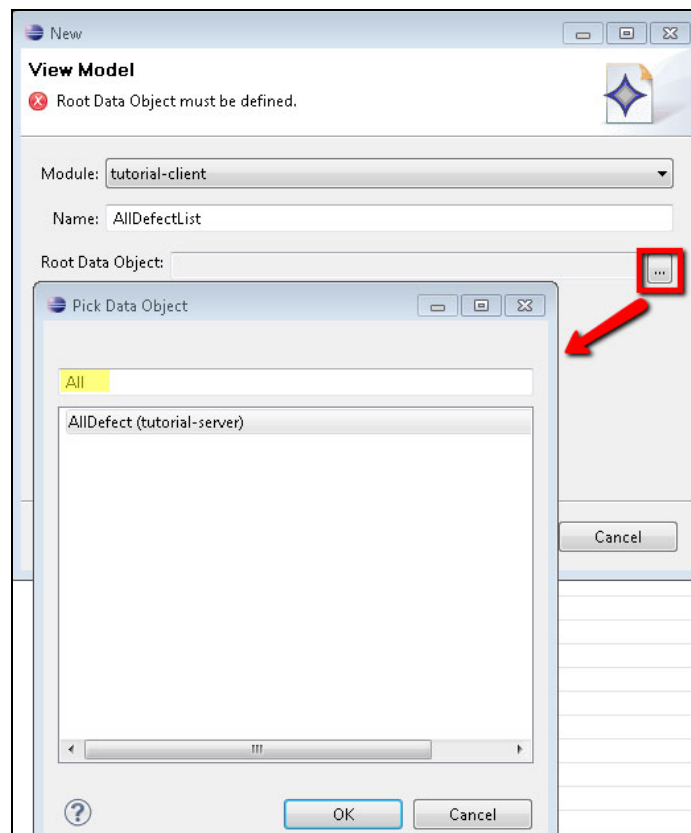
1. In the **CodeObjects Metadata** pane, right-click the **tutorial-client** module and choose **New** → **User Interface** → **View**.



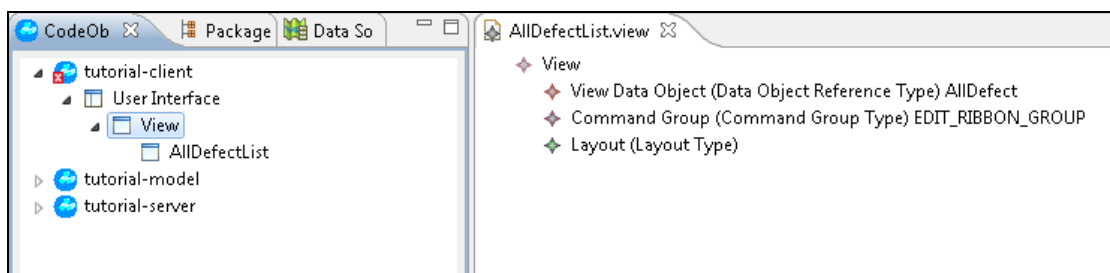
2. In the dialog box, set the options:

- a. **Module** Click the **List** arrow and choose **tutorial-client**.
- b. **Name** Enter **AllDefectList**.
- c. **Root Data Object** Click the **Pick** button, choose **AllDefect (tutorial-server)** and click **OK**.

This uses the **AllDefect** data object that you created earlier. You will create a very simple view that displays all of the fields of the data table to the user.



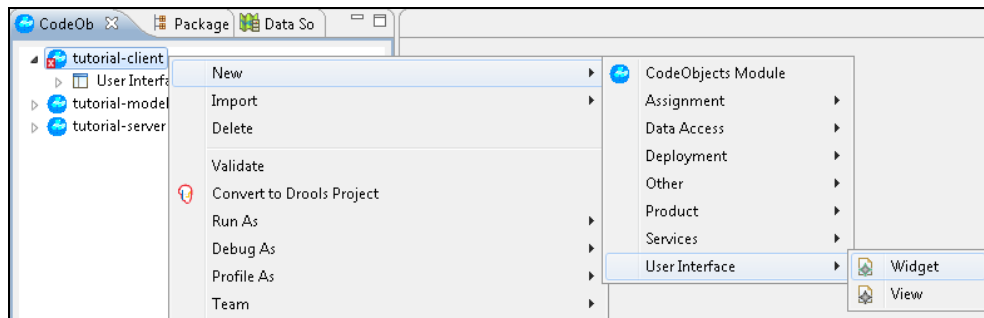
3. Click **Next**, review the settings, but leave them at their default values and click **Finish**. In the Metadata pane, the new view appears under **tutorial-client**.



Create a Widget

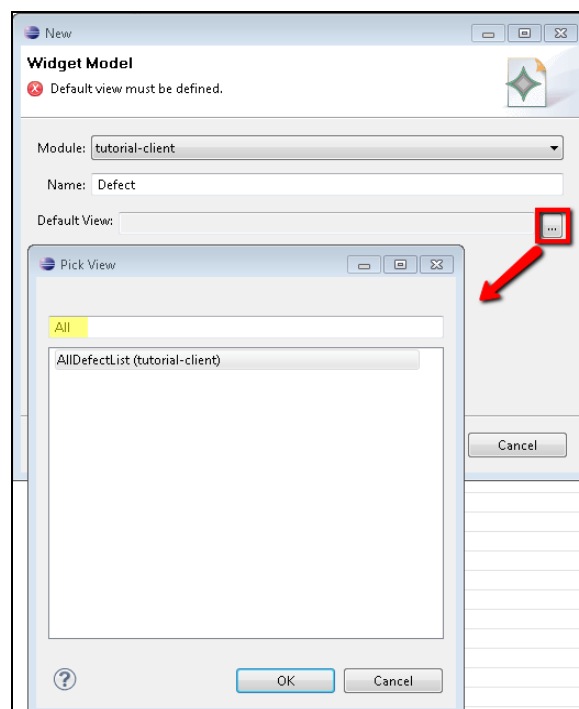
In this section, you will create the Widget metadata **Defect.widget** that acts as the container to include a list of user interface views.

1. Right-click the **tutorial-client** module and choose **New** → **User Interface** → **Widget**.



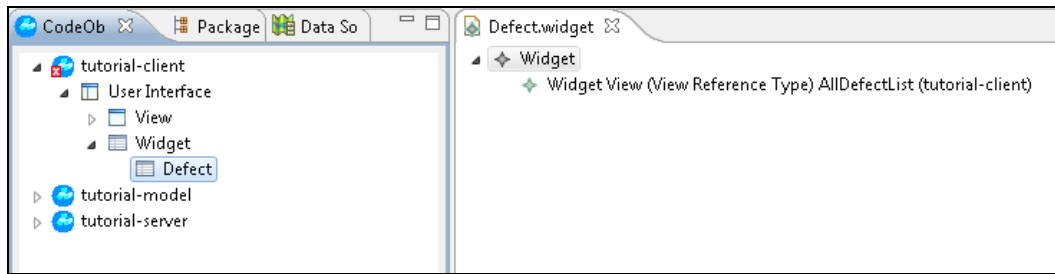
2. In the dialog box, set the options:
 - a. **Module** Click the **List** arrow and choose **tutorial-client**.
 - b. **Name** Enter **Defect**.
 - c. **Default View** Click the **Pick** button, choose **AllDefectList (tutorial-client)**, and click **OK**. Click **Finish**.

Note that the default view is the one that will be displayed by default when the user clicks the widget (tab).



Defining the User Interface Layer

In the Metadata pane, the widget appears under the **User Interface** node.



You have just created a basic user interface with all default settings. The purpose of this first tutorial is to work through the entire process of creating an application based on CodeObjects Platform. Later tutorials will demonstrate how to customize the user interface, but for now, you have been able to quickly and easily create a simple working interface.

Deploying the Whole Application

In this chapter, you will create the client-side deployment and deploy the whole application.

In this chapter, you will create the deployment metadata for the UI layer for what you have created so far. You will choose which UI views and widgets will be available in this application and set the user authorization for the UI. For example, certain users cannot access certain button or views.

You will be working with the **tutorial-client** module to save your metadata and deployment settings.

High Level Metadata in the Deployment Layer for Clients

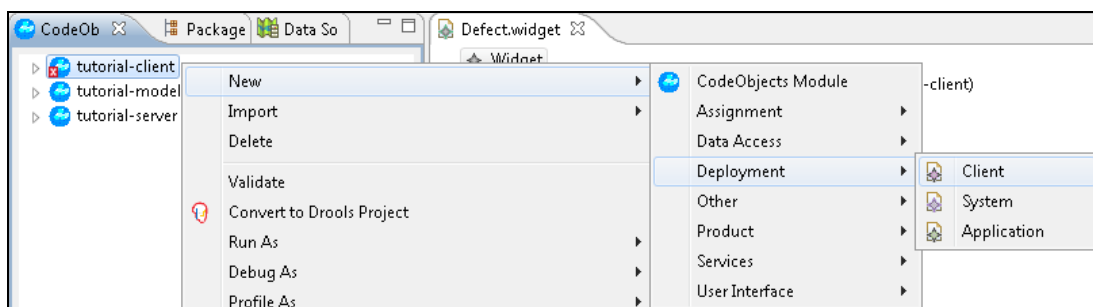
The high level metadata objects that you create in this chapter will appear in the **CodeObjects Metadata** pane as you create them:

- **Client** The client metadata defines what should be packaged together for the user interface; that is, it describes the deployment content for the user interface layer.

Create a Client (Metadata)

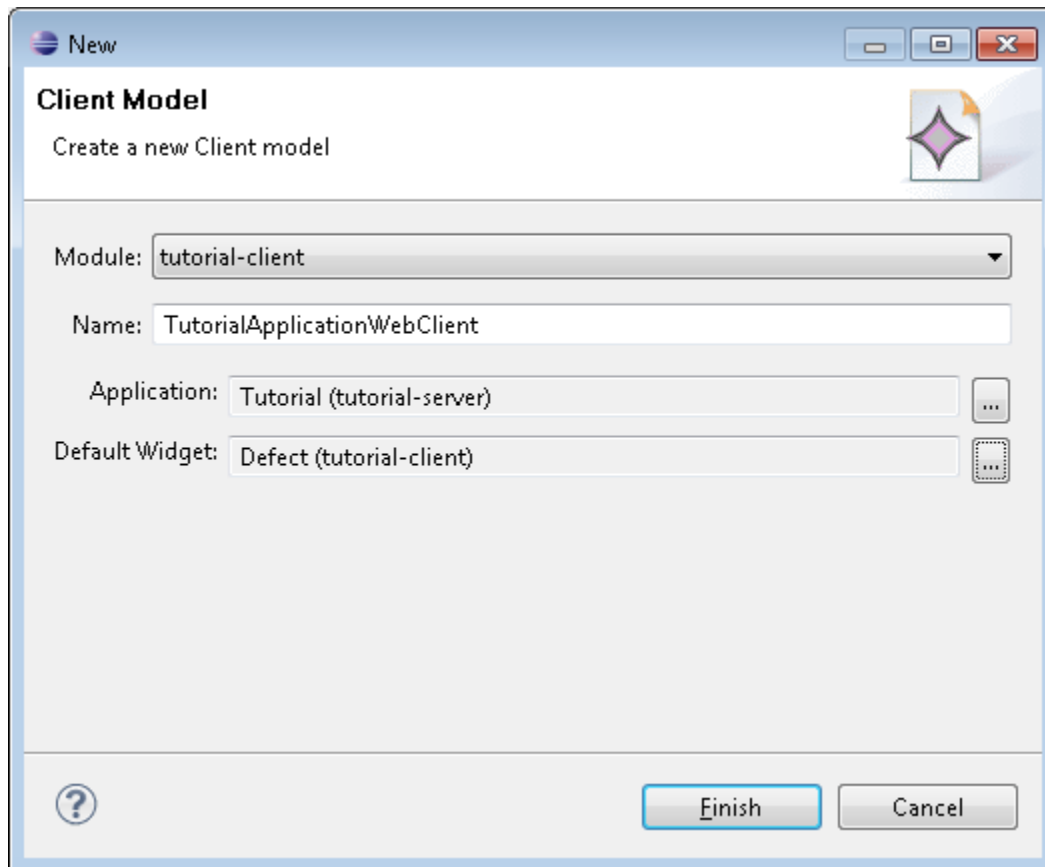
To create the Client metadata:

1. In the Metadata pane, right-click **tutorial-client** and choose **New** → **Deployment** → **Client**:



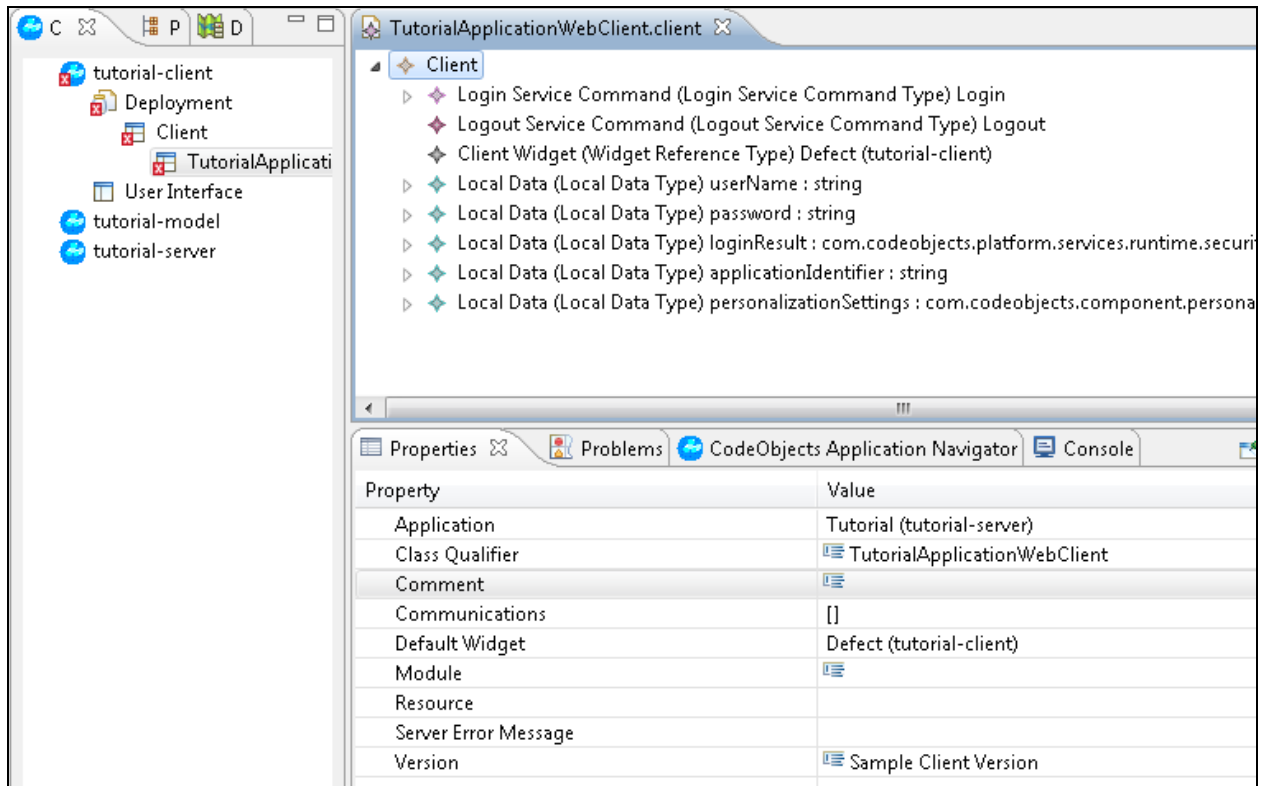
2. In the dialog box, set the options:

- a. **Module** Click the **List** arrow and choose **tutorial-client**.
- b. **Name** Enter **TutorialApplicationWebClient**.
- c. **Application** Click the **Pick** button and choose **Tutorial (tutorial-server)**.
- d. **Default Widget** Click the **Pick** button and choose **Defect (tutorial-client)**.
Click **Finish**.

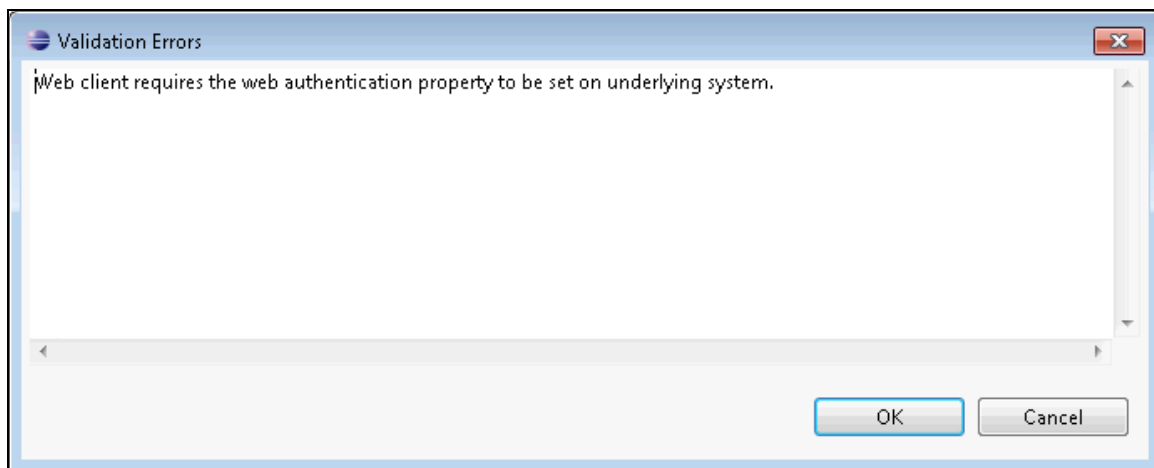


Deploying the Whole Application

The Metadata pane displays the **Client**, the Editor pane displays the **Client** nodes, and the Property pane displays the **Properties** to be set.

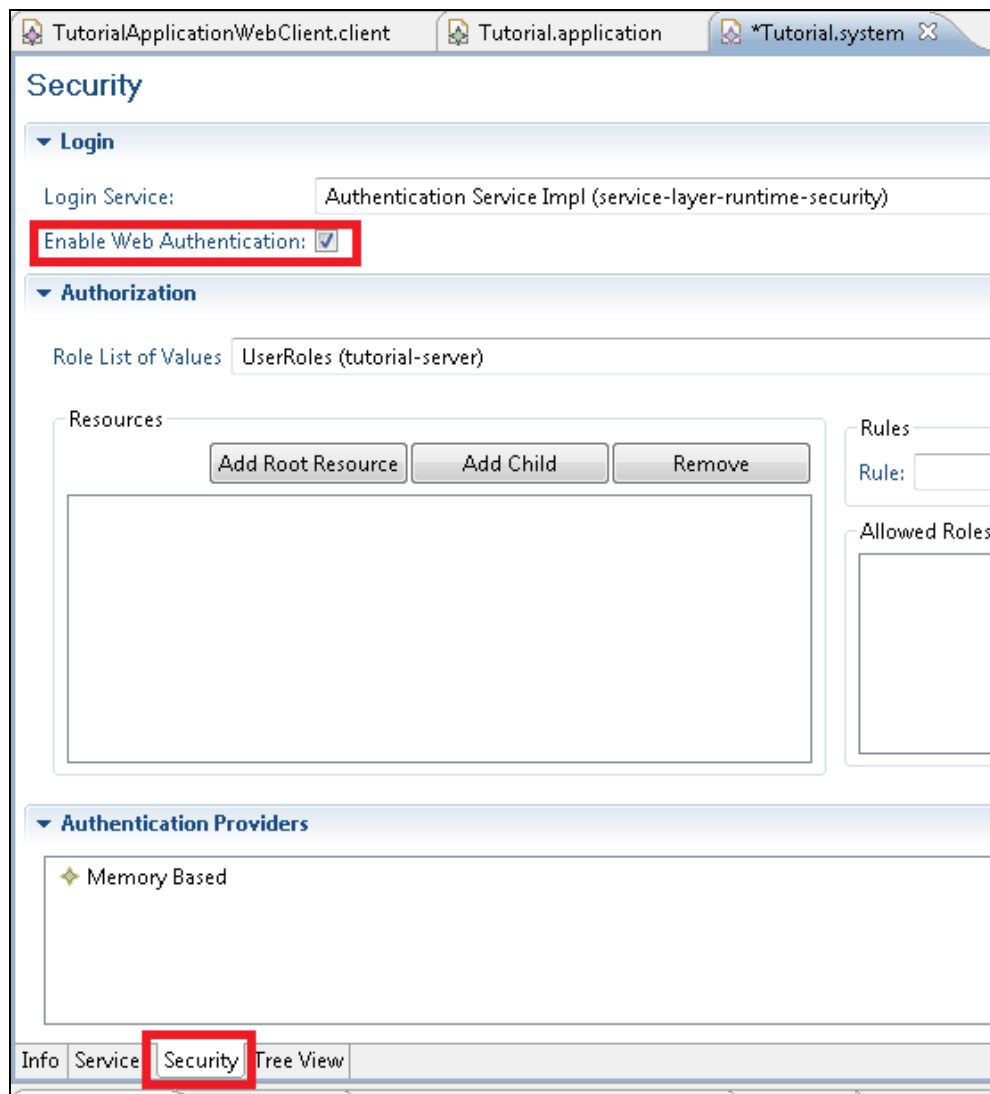


3. Note the validation errors, indicated by red X symbols  in the Metadata pane. Right-click the node and choose **View Validation Errors**.



In this case, the System metadata (Tutorial.system) does not set the Web Authentication property.

4. To fix this, in the Metadata pane, double-click **Tutorial.system**. (If the tab is not visible, from the menu click **tutorial-server** → **Deployment** → **System** → **Tutorial** to display it.)
5. In the Editor pane, click the **Tutorial.system** tab, under **Security** select the **Enable Web Authentication** check box, and save your metadata (press CTRL+S.)

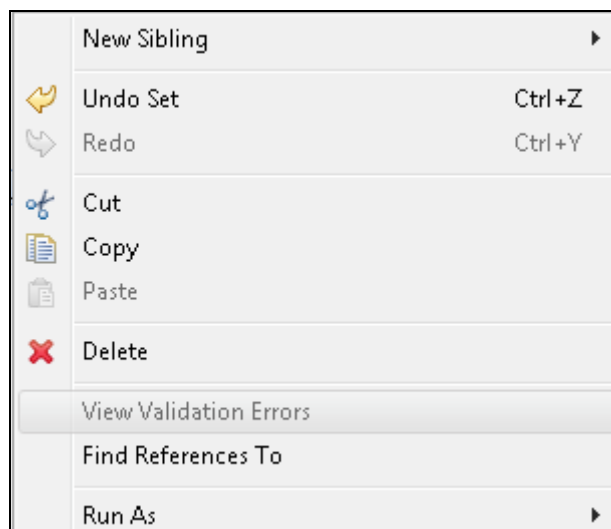


A Note on Errors in CORADS/Eclipse

Consistent with standard convention, errors in CORADS appear in the Metadata pane as red X symbols  located on each node with which an error has occurred.

To diagnose an error, you can double-click a node in the Metadata pane to display its metadata in the Editor pane. The erroneous metadata entry appears in red. Rest the mouse pointer on the entry to display a description (tooltip) of the error. Double-click a metadata node to display the associated properties in the Property pane so that you can fix the error.

Alternatively, you can right-click the node and choose **View Validation Errors**.

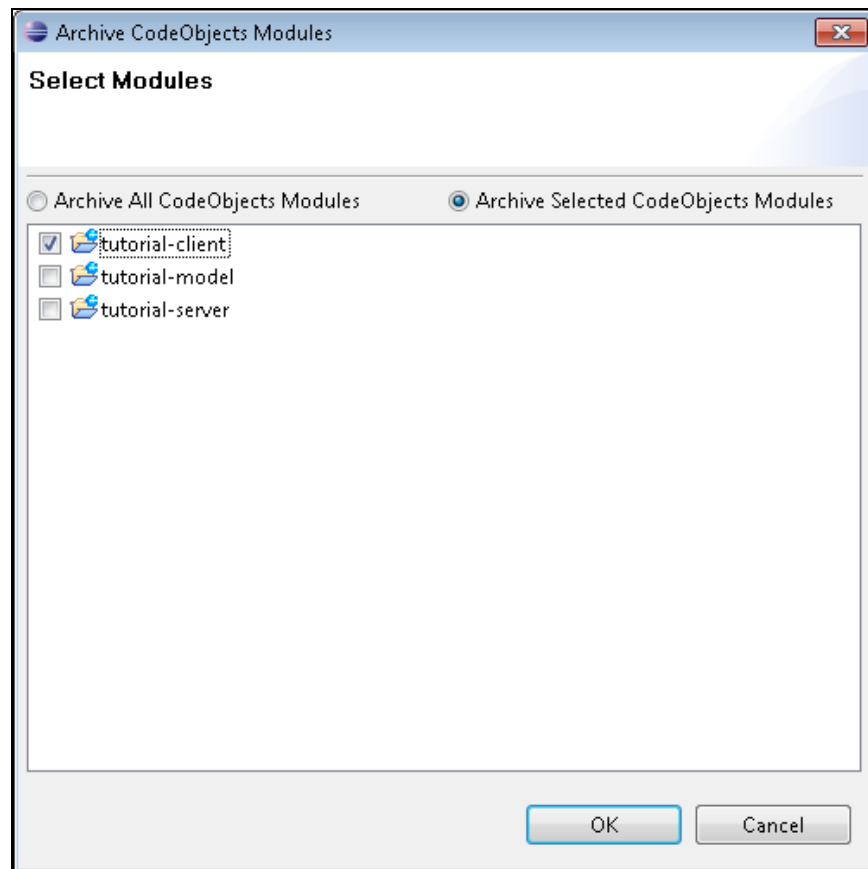


Prepare the Deployment for the Client Module

You have completed all the metadata construction for the client side. You can create the deployment file (Web Archive or WAR file) for the client. Because this client is marked as a web client, CORADS will generate a WAR file that contains both the client and the service that you previously specified.

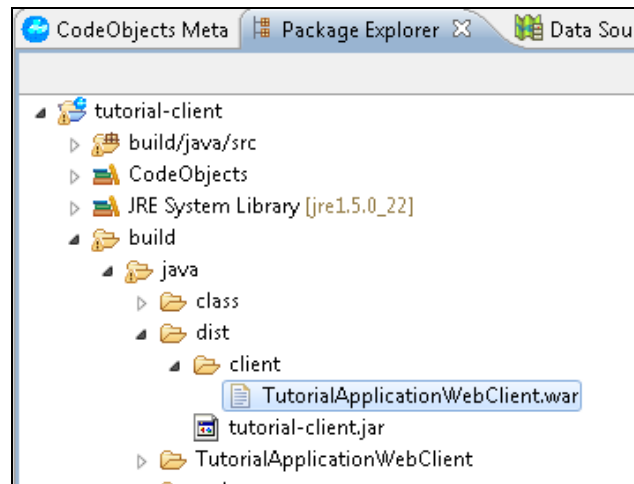
To generate the WAR file:

1. From the menu, click **CodeObjects** → **Archive Module**. Select the **tutorial-client** check box, clear the **tutorial-server** check box, and click **OK**. Note that because the **tutorial-client** module metadata references the other modules, CORADS includes the other modules automatically in the build, creating a single WAR file that includes all modules.



2. In the Metadata pane, click the **Package Explorer** tab. If it is not visible, click **Window** → **Show View** → **Package Explorer**.

3. Expand the tree through **tutorial-client** → **build** → **java** → **dist** → **client** to display the generated WAR file
TutorialApplicationWebClient.war



Deploy and Run the Whole Application

You have successfully built the initial base defect tracking web-based application. Deploy it to Tomcat and see the application in live operation.

NOTE

Before you deploy the new WAR file, if you have previously deployed the server side WAR file, you must stop Tomcat and delete the old WAR file and Tutorial Application folder before proceeding.

Copy the WAR file from:

**<CodeObjectsInstallRoot>\modules\tutorial\tutorial-client\
build\java\dist\client\TutorialApplicationWebClient.war**

to your Tomcat installation's **webapps** folder and start Tomcat.

(Instead of navigating through that source path, in the **Package Explorer** right-click the WAR file and choose **Copy**.)

After Tomcat has started, you can access the application at:

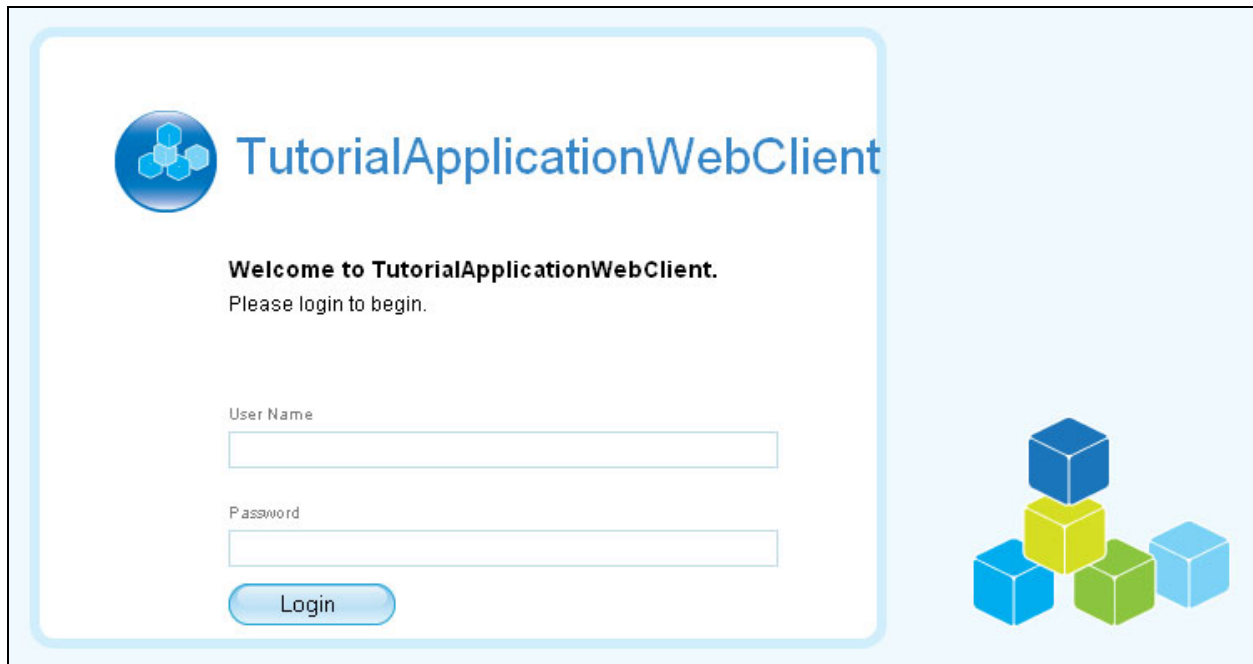
<http://localhost:8080/TutorialApplicationWebClient>

which should immediately redirect you to:

<http://localhost:8080/TutorialApplicationWebClient/spring/login>.

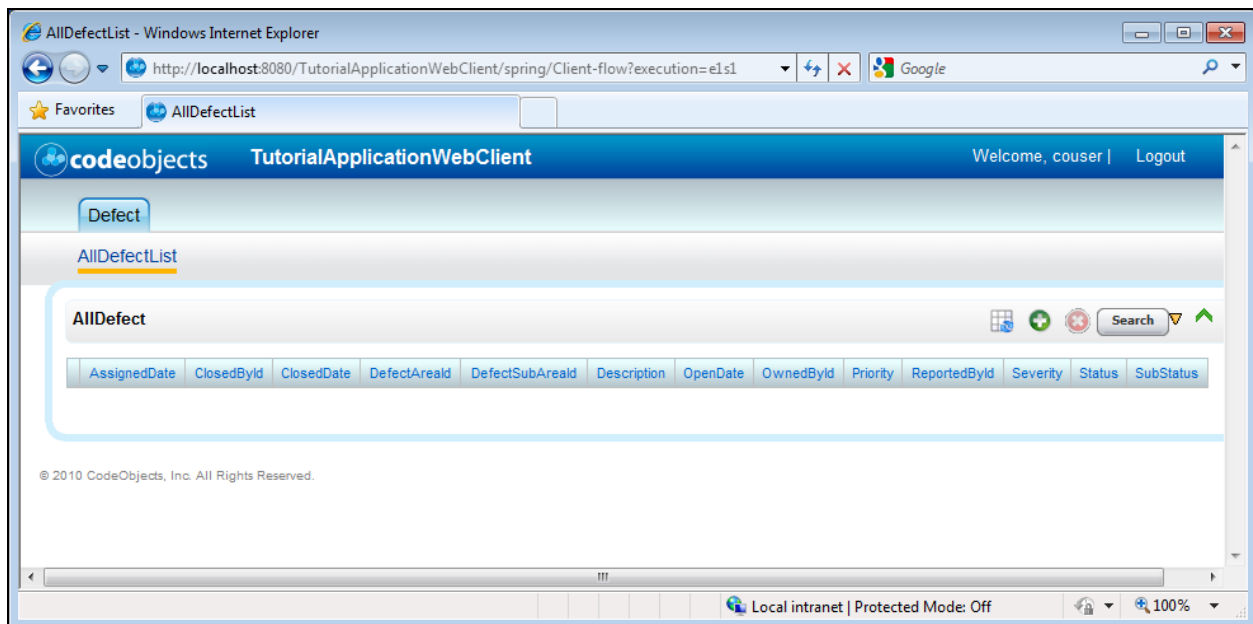
Deploying the Whole Application

The opening screen should appear:



Login as user **couser** (password **couser** . That is the user you set up earlier in the tutorial.

After you have logged in, the application screen appears.



Note that the tab corresponds to the widget name, **Defect**, and links on the Explorer tab and the view are named **AllDefectList**. The data is labeled **AllDefect**, where you can create new records or delete, search, and update them. Recall also that you did not customize your view or widget in any way, so the fields are displayed with their field names, rather than more descriptive names. (A later tutorial covers user interface customization.)

Create a Defect

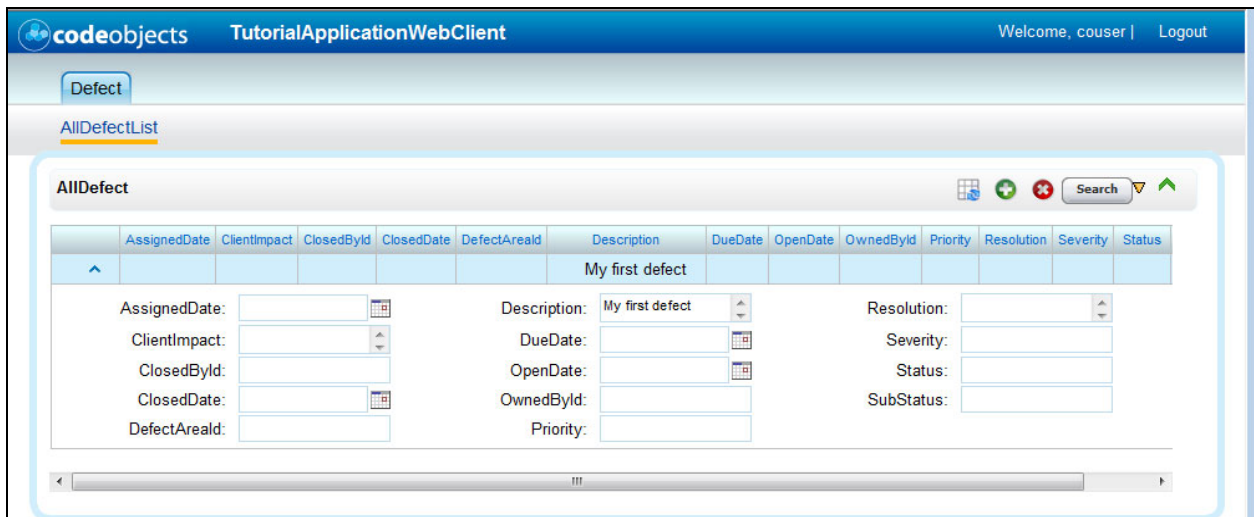
Use the defect tracker to create a defect report:

1. Click the **New** button  to create a defect.

This tutorial did not specify any rules to enforce, such as required fields or data integrity checks, so in the **Description** field, just enter **My first defect**.

2. Click the Update button  button to save the record.

3. The application should appear something like this:

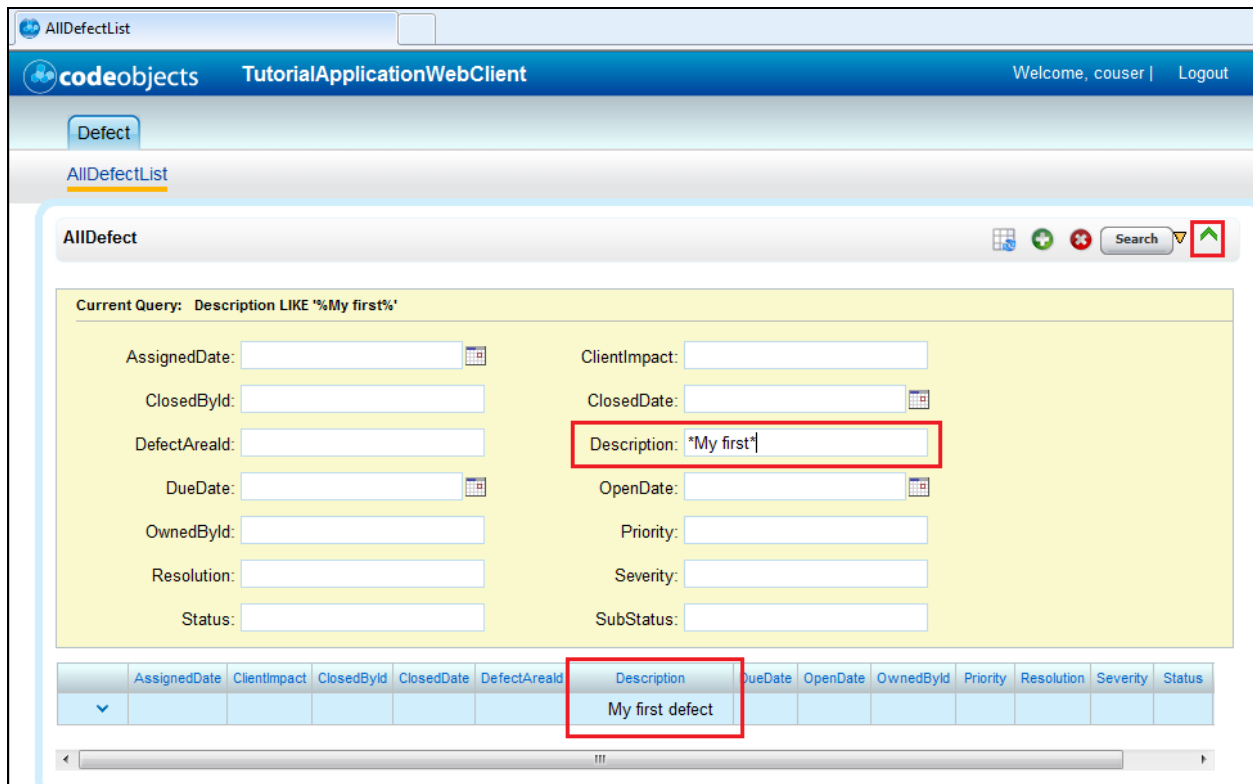


The screenshot shows the 'codeobjects TutorialApplicationWebClient' interface. At the top, there's a 'Defect' tab and a 'Welcome, couser | Logout' link. Below the tab is the 'AllDefectList' view. The view has a toolbar with icons for grid, new, delete, and search. The main area displays a table with the following columns: AssignedDate, ClientImpact, ClosedById, ClosedDate, DefectAreald, Description, DueDate, OpenDate, OwnedById, Priority, Resolution, Severity, and Status. The 'Description' column is highlighted, and the text 'My first defect' is entered in the corresponding field. Below the table, there are input fields for AssignedDate, ClientImpact, ClosedById, ClosedDate, DefectAreald, Description (with 'My first defect' entered), DueDate, OpenDate, OwnedById, Priority, Resolution, Severity, and SubStatus.

Search for a Defect

To search for the defect you just created, click the **Search** button.

Alternately, you can click the **Query** button  button to open the query panel. In the query **Description** field, enter ***My first*** which is partial text of the description that you used to when you first created the defect. When done, click the **Search** button, and the record should appear.

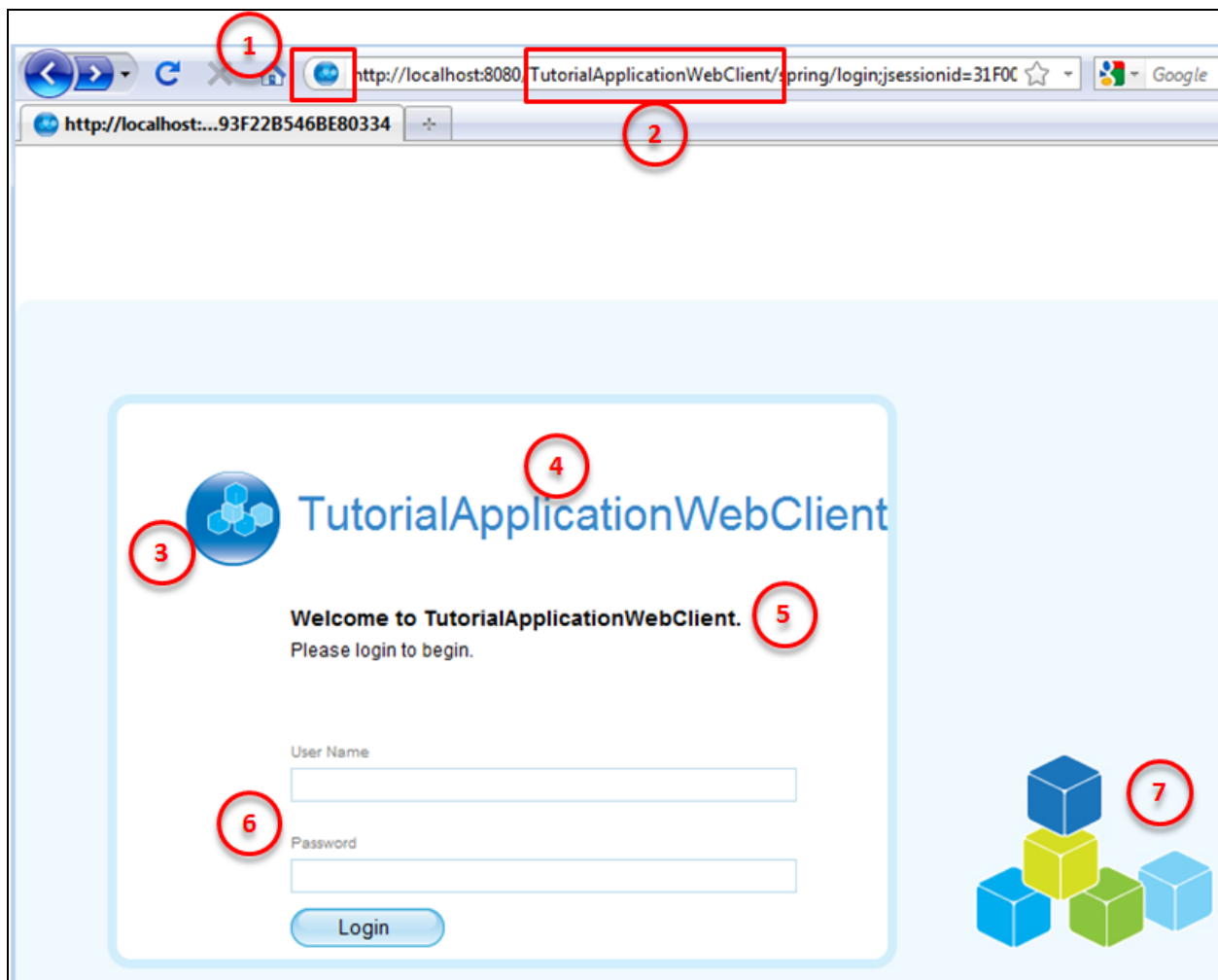


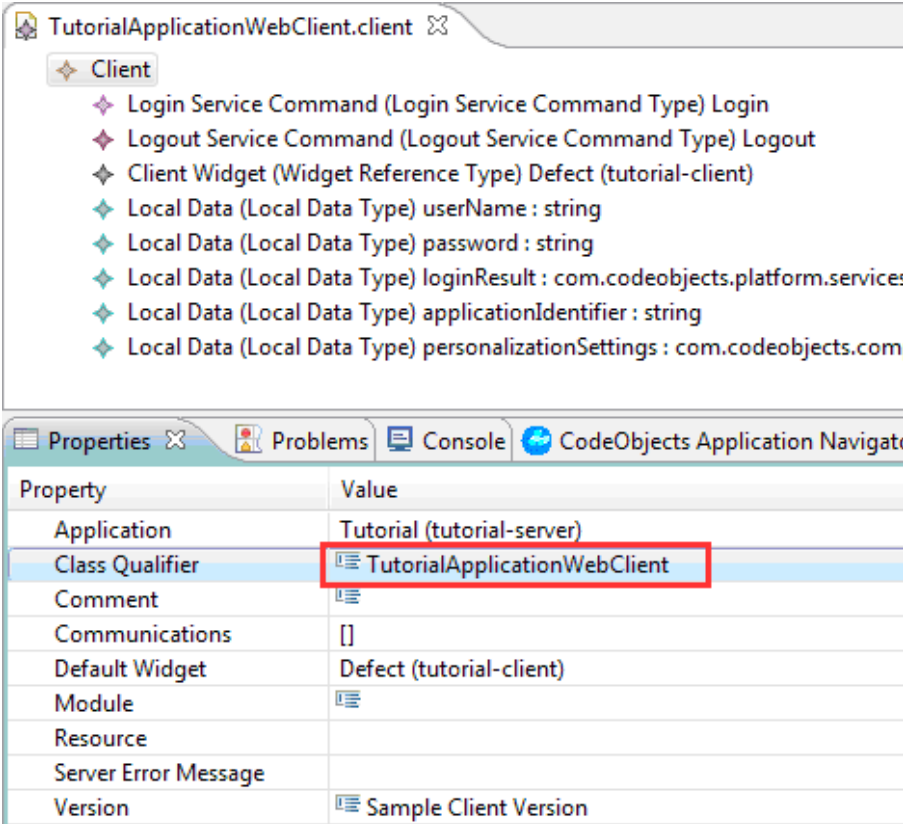
The screenshot shows the 'AllDefectList' application interface. At the top, there's a header with 'codeobjects' and 'TutorialApplicationWebClient'. Below that, a 'Defect' tab is selected. The main area is titled 'AllDefect' and contains a search form. The 'Current Query' is 'Description LIKE "%My first%"'. The search form has various input fields: AssignedDate, ClientImpact, ClosedById, ClosedDate, DefectAreaId, Description (containing '*My first*'), DueDate, OpenDate, OwnedById, Priority, Resolution, Severity, Status, and SubStatus. The 'Description' field is highlighted with a red box. Below the search form is a table with columns: AssignedDate, ClientImpact, ClosedById, ClosedDate, DefectAreaId, Description, DueDate, OpenDate, OwnedById, Priority, Resolution, Severity, and Status. The first row of the table has a dropdown arrow in the first column and the text 'My first defect' in the 'Description' column, which is also highlighted with a red box. At the top right of the search form, there are icons for grid, add, delete, search, and a green upward arrow icon.

Matching Metadata with Results

This chapter walks through features of the deployed application and how they match with the metadata that you created.

In review of the application that you just built, consider the source of each UI element, starting with the login page when the application starts.



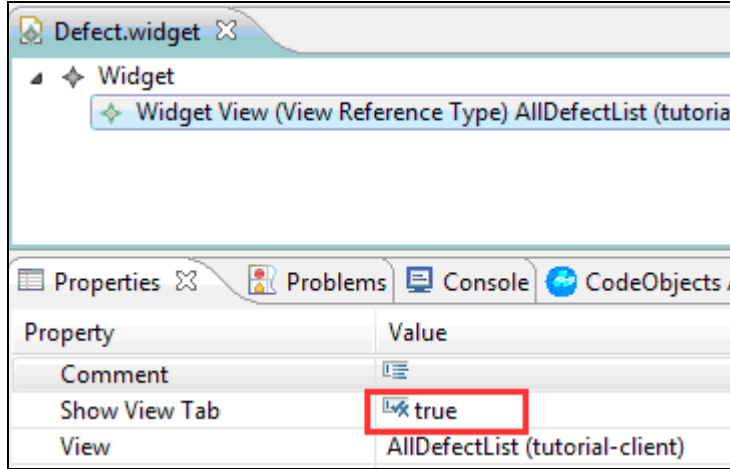
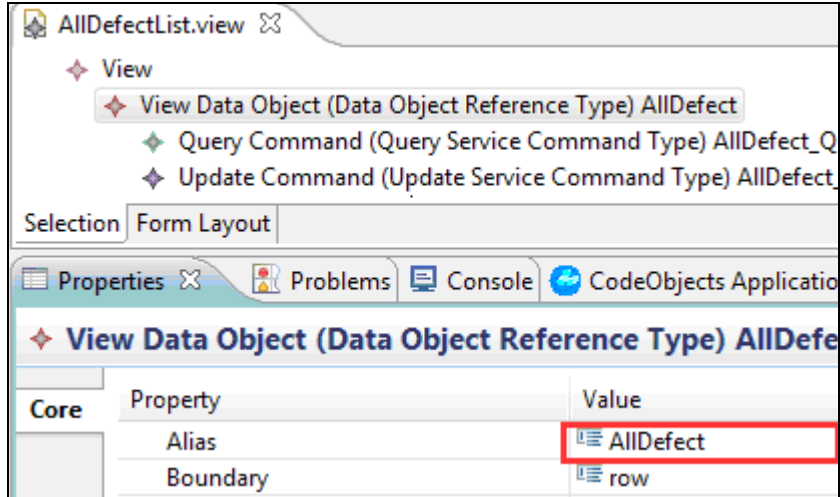
	UI Element	Description
1	Browser shortcut icon	This is included by default and can be replaced with your icon. A later tutorial shows how to replace this icon.
2	Application name in URL	This is from the WAR file name. This file name is determined by the property of Class Qualifier defined in the Client metadata. 
3	Application icon	This is included by default and can be replaced with another image. A later tutorial shows how to replace this.
4	Application name	This is the resource text defined for the client metadata. If there is no resource defined, by default the system uses the name of the metadata file (TutorialApplicationWebClient in our example).
5	Application name in the welcome message.	This is the resource text defined for the client metadata. If there is no resource defined, by default the system uses the name of the metadata file (TutorialApplicationWebClient in our example).
6	Login panel	Generated by the system automatically.
7	Application theme icon	This is included by default and can be replaced with any other icon. A later tutorial shows how to replace this.

Matching Metadata with Results

Next consider the main screen that appears after login.

The screenshot shows the main interface of the TutorialApplicationWebClient. It features a browser tab labeled 'AllDefectList' (1), a branding icon (2), and the application name 'TutorialApplicationWebClient' (3). The user is logged in as 'couser' (4) with a 'Logout' link (5). A 'Defect' button (6) and the 'AllDefectList' title (7) are visible. The main content area (8) contains a 'Current Query' section (14) with various search filters like 'AssignedDate', 'ClosedById', 'DefectAreald', 'DueDate', 'OwnedById', 'Resolution', 'Status', 'ClientImpact', 'ClosedDate', 'Description', 'OpenDate', 'Priority', 'Severity', and 'SubStatus'. A search button (9) and a list view icon (10) are also present. Below the filters is a table with columns for each field (15). The table contains one row of data, with the 'OpenDate' field highlighted by a red circle (16).

UI element	Description
1	AllDefectList in the browser tab
2	Branding icon
3	Application name
4	The welcome message with the user's login id.
5	Logout

6	Defect widget tab	This is the resource text defined in the widget. If there is no resource defined, by default, the system uses the name of the widget metadata. In the tutorial, this is the Defect widget.
7	AllDefectList view link	All the views included in the widget that have Show View Tab set to true will appear as a link here. The system uses the view's resource text if defined. Otherwise the system uses the metadata name. In this example, AllDefectList is the metadata name. 
8	AllDefect list control header	The system decides the title through certain rules. In brief, this name AllDefect comes from the Alias property of the View Data Object because no resource text was defined.  As a more detailed explanation of how the system decides about the control's header: (using AllDefect view as an example) The initial generated view metadata has created a List control, and this list control is linked to the data slice path "" (root data slice) in the AllDefect data object. This means that this list control is to show the root data slice from the data object AllDefect.

8

AllDefect list control header
(continued)

Test 1: If the given UI control has a Title defined, the system uses this title.

AllDefectList.view

View

View Data Object (Data Object Reference Type) AllDefect

Command Group (Command Group Type) EDIT_RIBBON_GROU

Layout (Layout Type)

Grid Layout (Data Slice Grid Layout Type) Rows: 1, Columns:

Data Slice Cell (Data Slice Grid Cell Type) Row: 0, Columndiv>

Data Slice (Data Slice UI Type) AllDefect

Control (List Control Type)

Selection

Form Layout

Properties

Problems

Console

CodeObjects Applicat

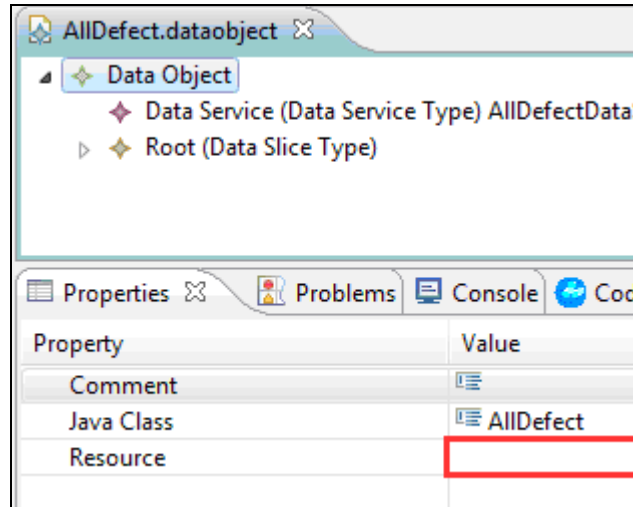
Control (List Control Type)

Core	Property	Value
	Comment	
	Disable Dirty Row Style	☐ false
	Disable Sort	☐ false
	Double Click Command	
	Number Of Rows	7
	Show Multi Select Check Box	☐ false
	Show Slider	☒ true
	Title	

8

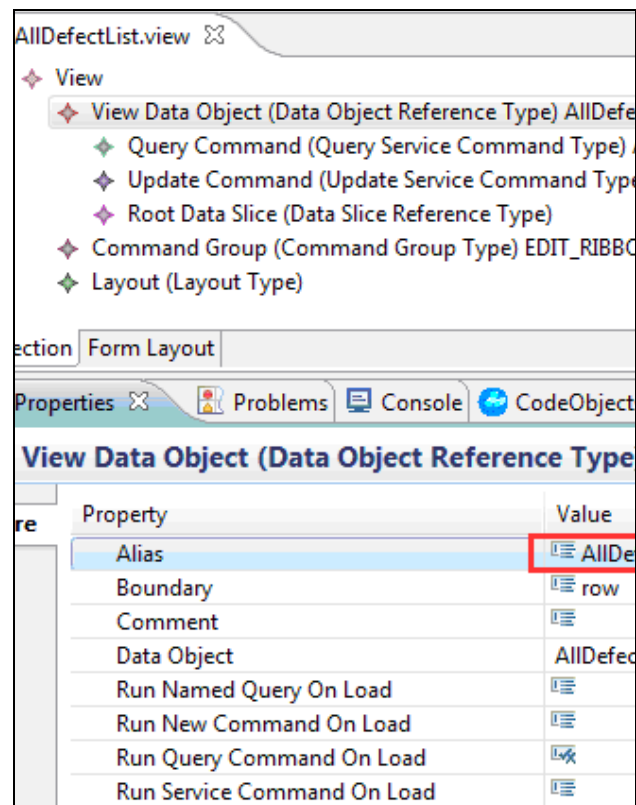
AllDefect list
control header
(continued)

Test 2: If the data slice to which this UI control is linked is a root data slice in the data object, and the data object has its Resource property set, use it.



If the data object does not have a Resource, use the underlying Entity's Resource if defined.

If none of the above has any Resource set, use the Alias name of the data object defined in view metadata.



8

AllDefect list
control header
(continued)

Test 3: If the data slice to which this UI control is linked is *not* the root data slice in the data object:

If the data slice's property Association Name Resource is set, use that.
If the Association Name Resource is not set, use the underlying Entity's Resource, if defined.

If none of the above has any Resource property set, use the property *Java Class* defined in the corresponding Data Slice in the Data Object.

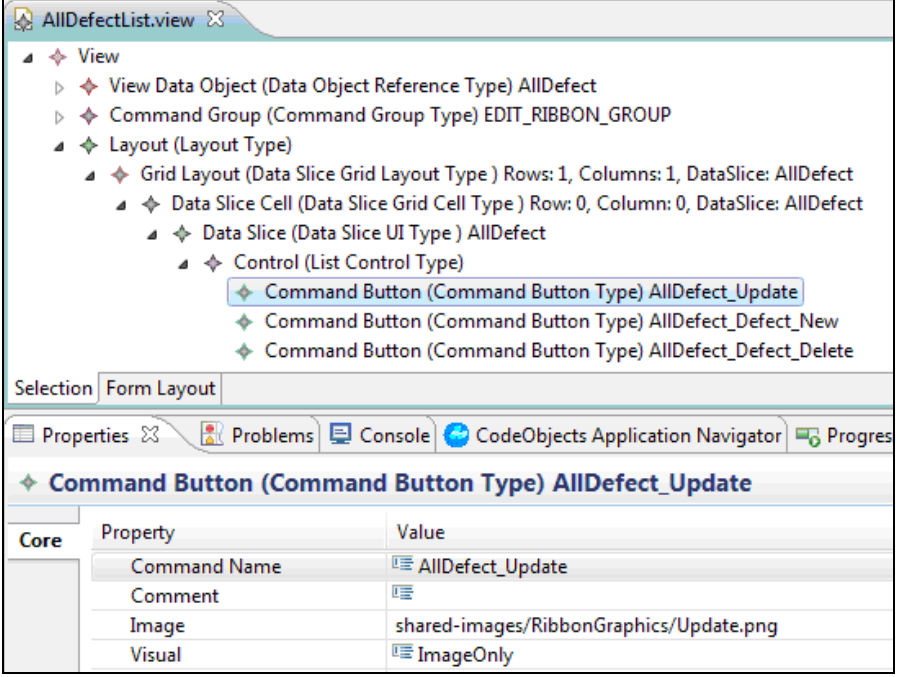
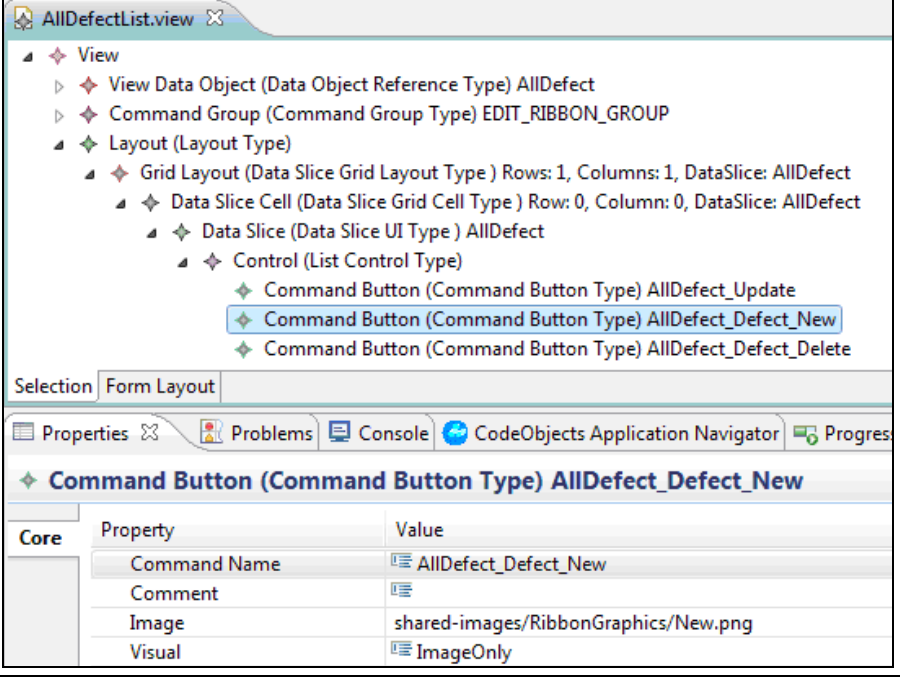
The screenshot displays the 'AllDefect.dataobject' tree and the 'Properties' window. The tree shows a hierarchy of virtual fields and child data slices. The 'Properties' window is open, showing the 'Java Class' property highlighted with a red box, set to 'DefectArea'.

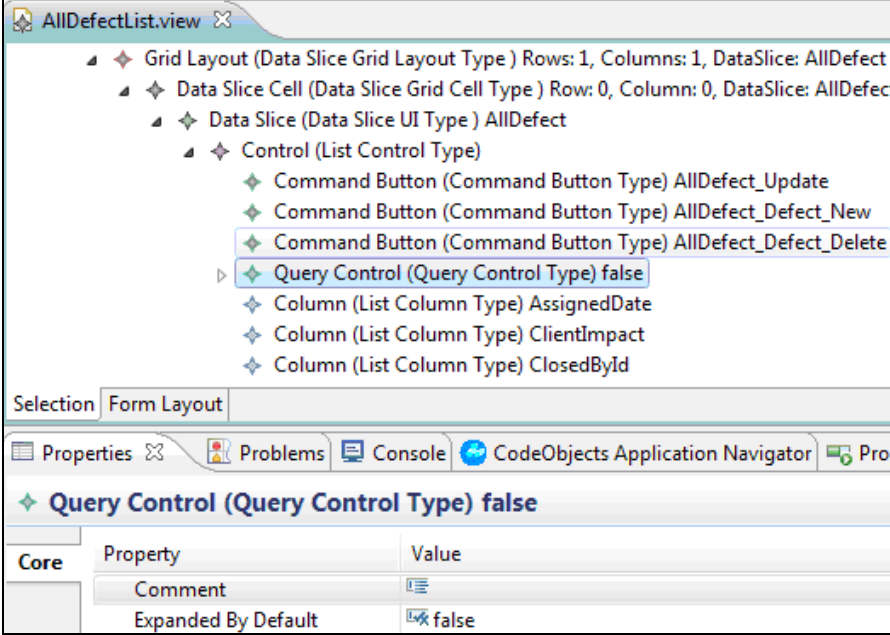
Tree Structure:

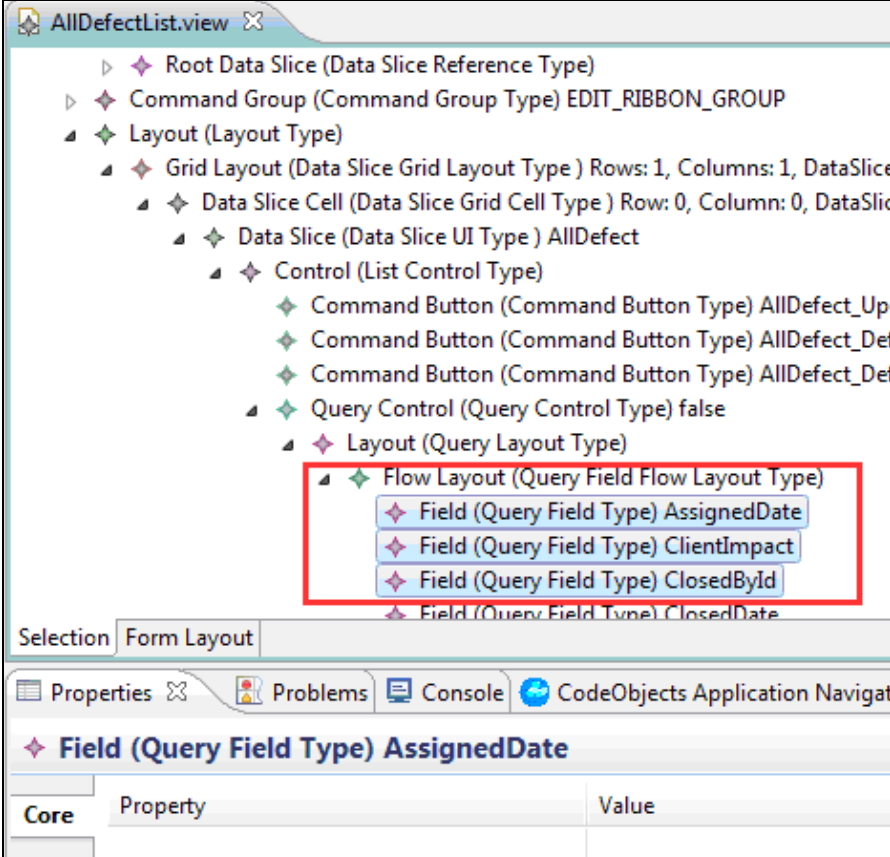
- Virtual Field (Data Slice Virtual Field Type) DefectAreaName
- Virtual Field (Data Slice Virtual Field Type) NotifyDefectAssignment
- Virtual Field (Data Slice Virtual Field Type) OwnedByEmployeeName
- Virtual Field (Data Slice Virtual Field Type) ClosedByEmployeeName
- Child (Data Slice Type) DefectArea, Entity: DefectArea (tutorial-mod)
- Child (Data Slice Type) OwnedByEmployee, Entity: Employee (comp)

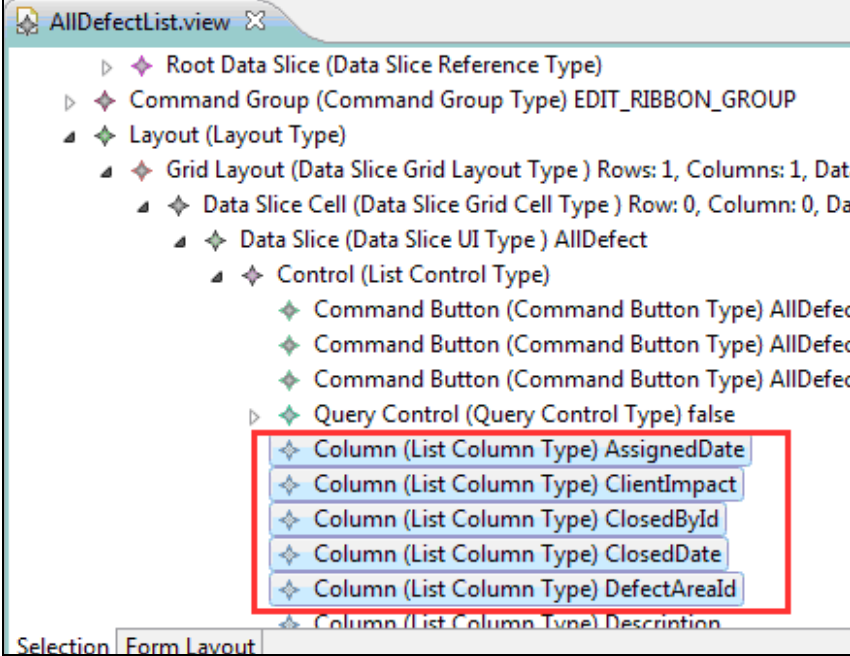
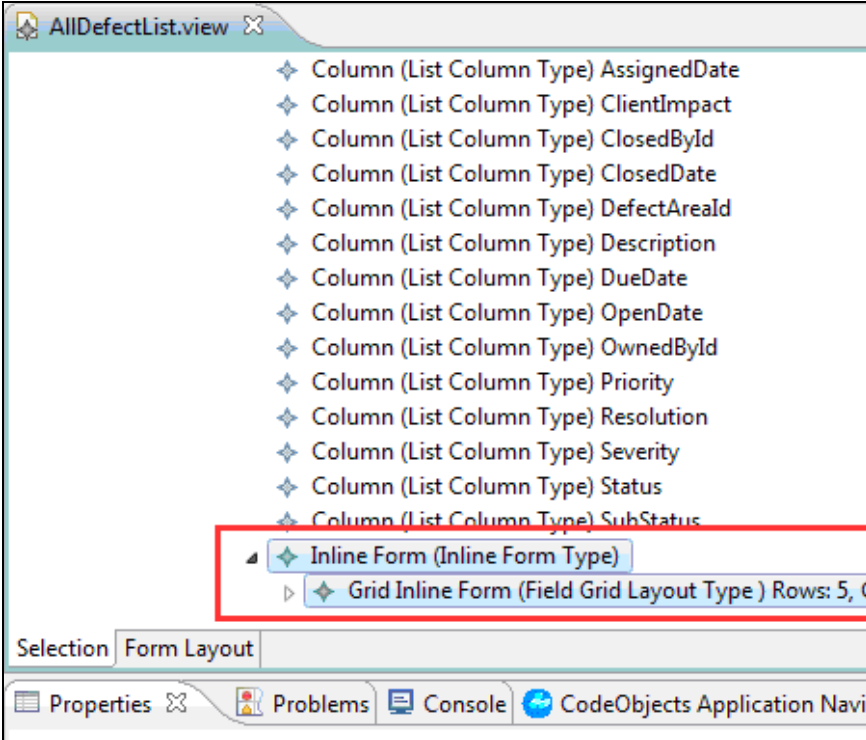
Properties Window:

Property	Value
Association	DefectArea
Association Name Resource	
Comment	
Enable Copy	
Entity	DefectArea (tutorial-model)
Java Class	DefectArea
Paging Size	1
Row Read Only Expression	
Slice Read Only Expression	

9	Update button	Generated by default from the View metadata created initially. The Update button is defined in the Layout section of the metadata.  The screenshot shows the 'AllDefectList.view' metadata editor. The 'Layout' section is expanded, showing a 'Grid Layout' with one row and one column. The 'Data Slice Cell' contains a 'Data Slice' of type 'UI', which contains a 'Control' of type 'List Control'. The 'List Control' contains three 'Command Button' elements: 'AllDefect_Update', 'AllDefect_Defect_New', and 'AllDefect_Defect_Delete'. The 'AllDefect_Update' button is selected. The 'Properties' pane shows the following details: <table border="1"> <thead> <tr> <th>Core</th> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td></td> <td>Command Name</td> <td>AllDefect_Update</td> </tr> <tr> <td></td> <td>Comment</td> <td></td> </tr> <tr> <td></td> <td>Image</td> <td>shared-images/RibbonGraphics/Update.png</td> </tr> <tr> <td></td> <td>Visual</td> <td>ImageOnly</td> </tr> </tbody> </table>	Core	Property	Value		Command Name	AllDefect_Update		Comment			Image	shared-images/RibbonGraphics/Update.png		Visual	ImageOnly
Core	Property	Value															
	Command Name	AllDefect_Update															
	Comment																
	Image	shared-images/RibbonGraphics/Update.png															
	Visual	ImageOnly															
10	New button	Generated by default from the view metadata created initially. The New button is defined in the Layout section of the metadata.  The screenshot shows the 'AllDefectList.view' metadata editor. The 'Layout' section is expanded, showing a 'Grid Layout' with one row and one column. The 'Data Slice Cell' contains a 'Data Slice' of type 'UI', which contains a 'Control' of type 'List Control'. The 'List Control' contains three 'Command Button' elements: 'AllDefect_Update', 'AllDefect_Defect_New', and 'AllDefect_Defect_Delete'. The 'AllDefect_Defect_New' button is selected. The 'Properties' pane shows the following details: <table border="1"> <thead> <tr> <th>Core</th> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td></td> <td>Command Name</td> <td>AllDefect_Defect_New</td> </tr> <tr> <td></td> <td>Comment</td> <td></td> </tr> <tr> <td></td> <td>Image</td> <td>shared-images/RibbonGraphics/New.png</td> </tr> <tr> <td></td> <td>Visual</td> <td>ImageOnly</td> </tr> </tbody> </table>	Core	Property	Value		Command Name	AllDefect_Defect_New		Comment			Image	shared-images/RibbonGraphics/New.png		Visual	ImageOnly
Core	Property	Value															
	Command Name	AllDefect_Defect_New															
	Comment																
	Image	shared-images/RibbonGraphics/New.png															
	Visual	ImageOnly															
11	Delete button	Generated by default from the view metadata created initially. The Delete button is defined in the Layout section of the metadata.															

12	Search button	Generated by default from the view metadata created initially. The Search button is determined by the existence of query panel metadata.  The screenshot displays the 'AllDefectList.view' hierarchy. The 'Query Control (Query Control Type) false' is selected. The Properties window below shows the following table: <table border="1"> <thead> <tr> <th>Core</th> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td></td> <td>Comment</td> <td></td> </tr> <tr> <td></td> <td>Expanded By Default</td> <td>false</td> </tr> </tbody> </table>	Core	Property	Value		Comment			Expanded By Default	false
Core	Property	Value									
	Comment										
	Expanded By Default	false									
13	Show/Hide button	Generated by default from the view metadata created initially. The buttons are determined by the existence of query panel metadata.									

14	Query panel	Generated by default from the view metadata created initially. All the fields defined within the Query Control metadata are the fields that appear in the query panel.  The screenshot displays the metadata tree for 'AllDefectList.view'. The tree structure is as follows: - Root Data Slice (Data Slice Reference Type) - Command Group (Command Group Type) EDIT_RIBBON_GROUP - Layout (Layout Type) - Grid Layout (Data Slice Grid Layout Type) Rows: 1, Columns: 1, DataSlice - Data Slice Cell (Data Slice Grid Cell Type) Row: 0, Column: 0, DataSlice - Data Slice (Data Slice UI Type) AllDefect - Control (List Control Type) - Command Button (Command Button Type) AllDefect_Up - Command Button (Command Button Type) AllDefect_De - Command Button (Command Button Type) AllDefect_De - Query Control (Query Control Type) false - Layout (Query Layout Type) - Flow Layout (Query Field Flow Layout Type) - Field (Query Field Type) AssignedDate - Field (Query Field Type) ClientImpact - Field (Query Field Type) ClosedById - Field (Query Field Type) ClosedDate The 'Field (Query Field Type) AssignedDate' is selected, and the Properties panel below shows the following table: <table><thead><tr><th>Core</th><th>Property</th><th>Value</th></tr></thead><tbody><tr><td></td><td></td><td></td></tr></tbody></table>	Core	Property	Value			
Core	Property	Value						

15	List control column headers	Generated by default from the view metadata created initially. The sequence defined in the metadata determines the order in the list control. 
16	List control inline form	Generated by default from the view metadata created initially. Inline Form metadata within List control determines the entire layout. 

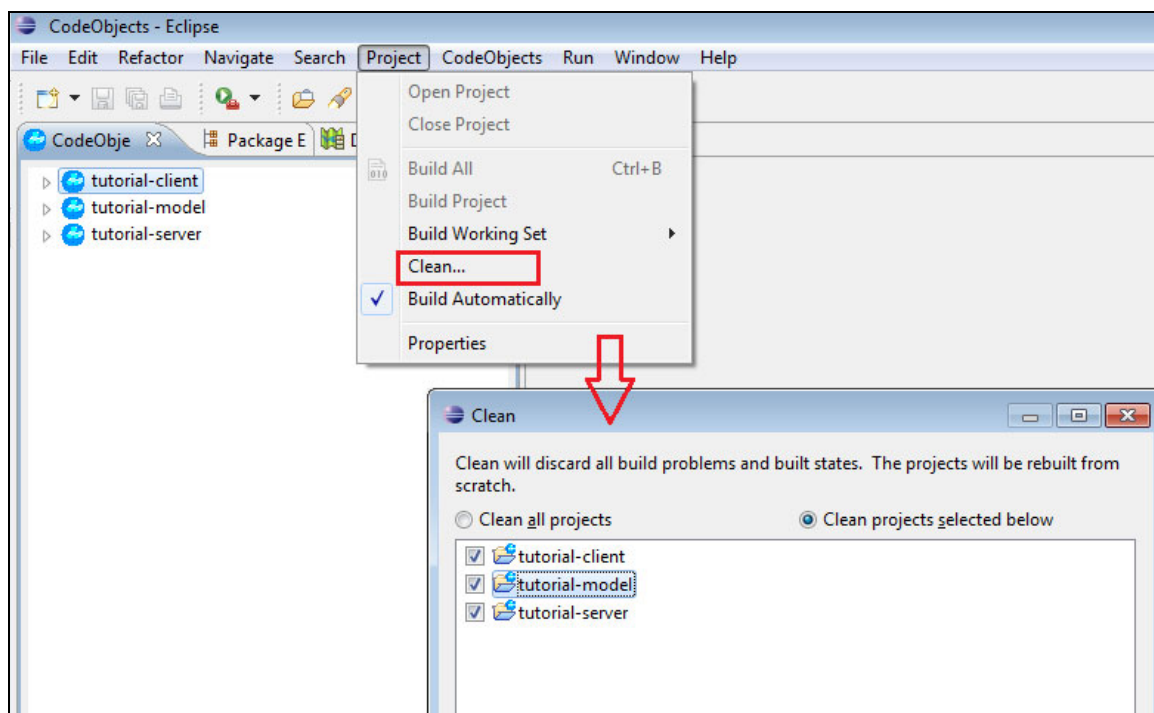
Checking the Application into Source Control

The final step in this tutorial is to check your files into source control.

Before Checking In

From the menu, click **Project** → **Clean** to remove all build folders (the folders that are created at each build cycle) so that the build artifacts are not checked in accidentally. This is useful because:

- It minimizes the number of files/folders that you check in.
- The build files will be re-generated at the next build, so there is no need to check them in.



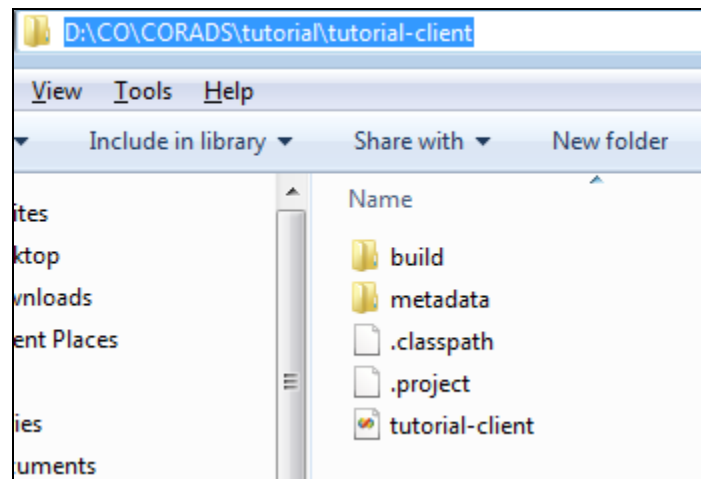
Files to Check In

Use the following guidelines when checking in files:

- Check in any source code or any files that you created.
- Do *not* check in:
 - Anything that is generated, including the interim metadata files, and binaries.
 - Anything that is related to your Eclipse environment.

Check in Your Files

For this tutorial, you created three modules, thus for each module, you must check in the files that you created. For example, for the **tutorial-client** module, the file folder looks like the following image before you run **Project** → **Clean**:



For the **tutorial-client** module, check in the files per the following table:

Location	Description	Check in?
Folder build	Generated metadata files, source files, binary files, and so on. This folder will be removed after you run Project → Clean.	No
Folder metadata	Metadata files that you created.	Yes
.classpath	Used by Eclipse.	No
.project	Used by Eclipse.	No
tutorial-client.module	Module file.	Yes

Follow the same check in process for the other modules: **tutorial-model** and **tutorial-server**.

Previewing the Complete Application

In this chapter, you will load the completed tutorial files and run them so that you can see what the final application looks like.

Now that you have seen the basic application in action, you might want to look ahead to the completed application—the files that you created in this tutorial *plus* all of the additions and adjustments that appear in the later tutorials.

We still recommend that you work through the complete tutorial set, but knowing where the tutorials are headed will help you understand them and build your skills more quickly.

NOTE

The finished tutorials involve a separate set of modules from the main tutorial set. They do not interact with the original set.

In the CodeObjects Metadata pane, the three finished-version modules display the prefix *finished*. As you continue with further tutorials, be aware that the original and finished sets appear in parallel. We recommend that you leave the finished module trees in a collapsed state to avoid confusion, except when you are using them to compare with the ongoing metadata structure.

Running the Finished Tutorial Files

Begin by importing the modules of the finished project. To load the finished files in CORADS/Eclipse:

1. Within the **CodeObjects Metadata** pane, right-click and choose **Import**. Import the following projects/modules:

finished-tutorial-model
finished-tutorial-client
finished-tutorial-server

2. Build the project: from the menu click **CodeObjects** → **Archive Modules** and choose the module **finished-tutorial-client**. Click **OK** to build the WAR file.
3. Stop Tomcat. Delete the old WAR file that you built:
TutorialApplicationWebClient.war
Also delete the folder that Tomcat created:
TutorialApplicationWebClient.
4. Copy the new WAR file:
finished-tutorial-client\build\java\dist\client\TutorialApplicationWebClient.war
to your installation's **webapps** folder and restart Tomcat.
5. Display the application by loading the following URL into your browser (assuming you are using a local Tomcat test server):
<http://localhost:8080/TutorialApplicationWebClient>
6. Log in to the opening screen as user **couser** (password: **couser**).

Examine the finished version.

We will discuss all the visible changes and more in future tutorials.

Summary

In this chapter, you will review the contents of this tutorial, including the process and the files that you created in CORADS.

Congratulations! You have developed a complete database-driven application, from start to finish. Your application includes a browser-based interface for users to log in securely and an attractive interface where users can add, update, retrieve, and display defect records. Consider that *in a brief period, you were able to assemble a complete application* that would have taken much longer using conventional programming methods.

Now that you have seen the basic process from start to finish, let's review the steps.

In this tutorial, you created:

- **Modules** Eclipse-based projects to contain your application; modules are not part of the MDD model, rather they are convenient ways to group your files.
- **Metadata** XML-based files that contain the specifications of an application.

Recall from the **Getting Started** guide that the CodeObjects Platform MDD model of application development consisted of layers of metadata, rather than collections of code. CORADS is the Eclipse-based toolset for building applications with CodeObjects Platform.

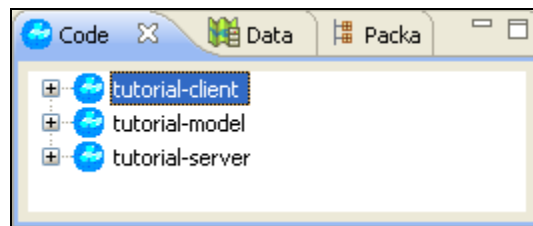
Application layers
Data Layer
Services Layer / Business Logic
Client Layer / User Interface: Views and Widgets
Packaging and Deployment Security: user authentication and granular authorization

Understanding the Module Structure

You created three modules for this application. Although you can create any structure that you want, the module structure that you used is a common and useful one with one module each for:

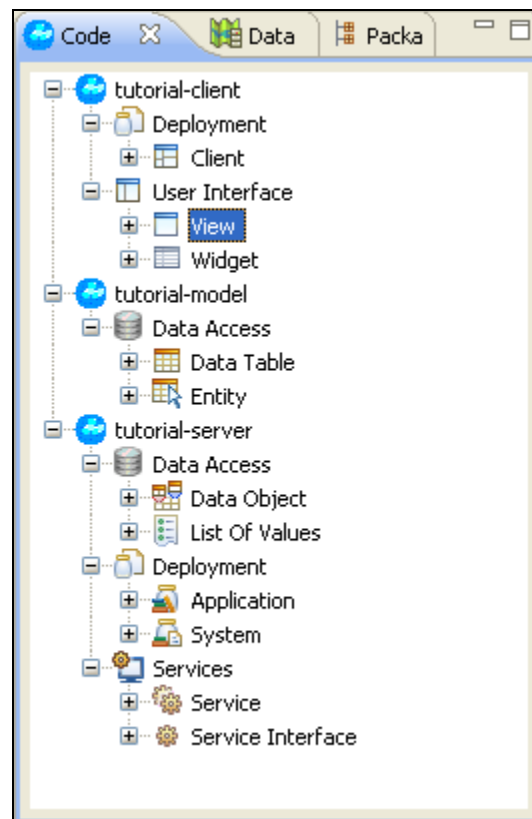
The data model	The database specification and the logical structure of the data—the Data layer
The application's server	Services to be provided to the user plus deployment metadata for the server-side—the Services layer
The application's client specifications	The user interface plus deployment metadata for the client-side—the Client layer

More complex applications can include other modules, but for learning purposes, this is a useful and practical organization. In CORADS/Eclipse, modules appear in the **CodeObjects Metadata** pane as top-level nodes.



Understanding the Metadata and How to Work with It

Within the modules, you created the metadata for each layer. Metadata is hierarchical, with one or two top-level metadata nodes for each layer containing a tree of child metadata nodes. The top-level nodes for each layer appear in the **CodeObjects Metadata** pane. The **CodeObjects Metadata** pane is a tree viewer that uses the same conventions as the Eclipse **Package Explorer** pane.

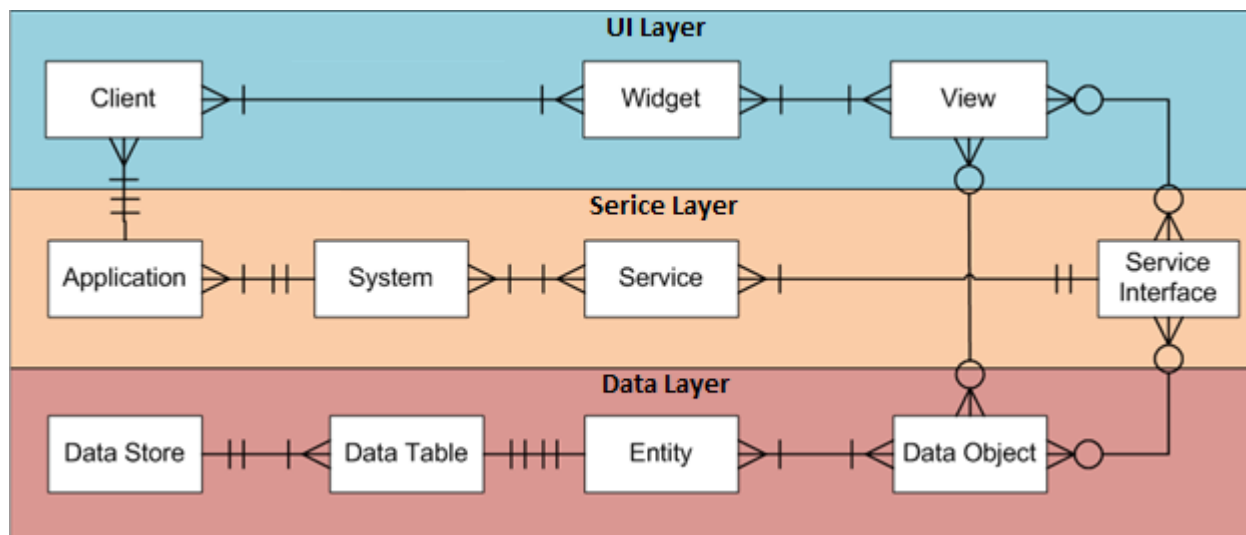


The top-level metadata nodes are:

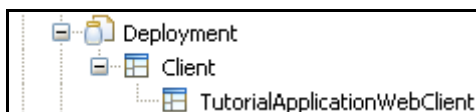
- For the **Data Access** layer Data Tables, Entities, Data Objects, and List Of Values
- For the **Services** layer Service Interfaces and Services
- For the **User Interface** layer Views and Widgets
- For the **Deployment** layer Client, Application and System

Summary

You created each of these high-level nodes in this tutorial. The following diagram displays the layers and the top-level metadata nodes at each layer.



When you open a top-level node type, in the **CodeObjects Metadata** pane, you see your own metadata nodes. In the following example, **TutorialApplicationWebClient** is a Client metadata node that you created in the tutorial, part of the deployment layer.



Double-click a node in the **CodeObjects Metadata** pane to open the tree in the Editor pane, so that you can see the child nodes. There are typically multiple levels of children.

Each metadata node contains attached properties, including the top-level nodes.

To see the properties of the top-level node

In the Metadata pane, double-click the node, and then in the Editor pane, click the top-most node of the tree. In the image below, that is the **Client** node.

To see the properties of lower-level nodes

Click them in the metadata tree.

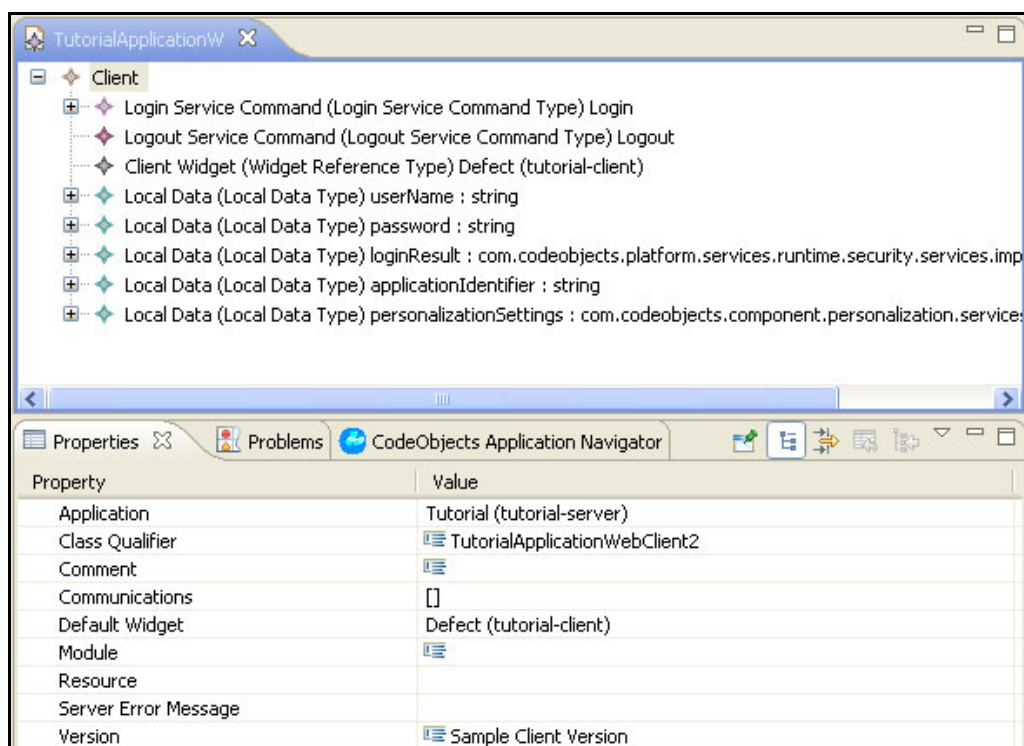
To open metadata nodes in the tree editor

Click the plus icon  beside the node (or the angle icon  depending on your system.)

To change the properties of a node

Simply change the values in the Property pane, below the Editor pane.

Summary



The top-level metadata nodes are:

- **Data Access** layer
 - Data Tables – field definitions of data tables imported from your database.
 - Entities – based on data tables, but encompassing business logic and system-wide rules for data validation.
 - Data Objects – scope of your work data on which you plan to operate, and this is used to generate data services.
- **Services** layer
 - Service Interfaces – the interface of the service.
 - Services – the implementation of the service interface.
- **User Interface** layer
 - Views – data displays (correspond to pages within tabs)
 - Widgets – a collection of views (correspond to tabs on the user interface)
- **Deployment** layer
 - Client
 - Application
 - System

In addition, you built other types of data nodes within the tutorial. Later tutorials provide additional information about specifying metadata.

We hope you found this first tutorial useful. We recommend that you perform the other CORADS tutorials and consult the User Guide.