

codeobjects



CORADS™

CodeObjects Rapid Application Development Studio

Tutorial 3

Data Management

CodeObjects, Inc.

Tutorial 3: Data Management

© 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 and Prerequisites	1
Description of the Tutorial Application	1
Before you Begin: Environment Setup.....	2
Adding More Data Tables	3
Working with New Data	3
The Procedure	4
1. Create the DefectArea Data Table by Importing from the Database.....	4
2. Create an Entity for the DefectArea Data Table	5
3. Create a Required Field in the DefectArea Entity	6
4. Define a Data Object for DefectArea	7
5. Create a New Data Object for the Employee Records	8
6. Add Data Objects to the System Metadata	9
Defining Relationships Between Entities.....	10
Overview of Entity Relationships	10
The Procedure	12
1. Define the Link Key to DefectArea in the Defect Entity	12
2. Define the Association Between the Defect Entity and the DefectArea Entity.....	15
3. Define the One-to-Many Association from the DefectArea Entity to the Defect Entity	17
Creating New Widgets and Views	19
Review of Widgets and Views.....	19
The Procedure	19

1. Create the View for Defect Areas	19
2. Create the View for Employee Data.....	23
3. Create a New Widget	26
4. Add the New Widget to the Client	27
Test the Result	28
Setting up Virtual Fields and Dynamic Lists Of Values (DLOVs)	30
Virtual Fields	30
Dynamic Lists	31
The Procedure	31
1. Add a Data Slice to the Data Object	31
2. Add a Virtual Field to the Data Slice	32
3. Configure the Data Source for a Dynamic List of Values	33
4. Create the Dynamic List Metadata	36
5. Change the UI Control to Use the Dynamic List.....	39
Test the Result	42
Review	43

Introduction and Prerequisites

*This tutorial, which continues from **Tutorial 2**, demonstrates additional techniques for working with data: multiple data tables, managing linked data, and virtual fields.*

*We assume that you've completed the previous tutorials, and that you have the files available. This tutorial continues to use the files that you created in **Tutorial 1: Fundamentals** and updated in **Tutorial 2: User Interface**. Note that you can complete **Tutorial 3** even if you did not perform the interface updates in **Tutorial 2**—the images presented here may just look a little different from what you see on your system.*

Description of the Tutorial Application

This tutorial application is a simple defect system for tracking bug reports in software development projects.

In the previous tutorials, the system used only one table, but in this tutorial, you will update the application to include three tables:

Defect Area	This contains names and descriptions of areas (categories) of defects. For example, you might include areas for Installation, Upgrading, Creating Data, Reporting, and Searching.
Defect	You built this in Tutorial 1. It contains a record for each specific defect/bug that is being tracked. It includes such data as a description of the problem, severity, priority, to whom the defect is assigned, and so on. Each defect record contains a defect area id, which is a foreign key field back to the defect area associated with the defect.
Employees	You will build this table from standard components that are pre-built by CodeObjects. This table will track the employees who create or update the defect records.

With your additional tables, your tutorial application will be able to support the following operations:

- Developers can define areas (categories) of defects to be tracked.
- Testers can log defects/bugs, assign them to areas, describe the defects, and assign the defects to specific developers.
- Developers and testers can update the system when the problems are addressed.

Before you Begin: Environment Setup

In assuming that you've completed **Tutorial 2**, we also assume that your environment is set up, and everything is installed correctly so that you are able to use CORADS to build, deploy, and test the basic application.

You are going to update your application, rebuild, and re-deploy the defect tracker application.

NOTE

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

Adding More Data Tables

In this chapter, you will define the metadata associated with the additional data that you need for your defect tracking application. The additional data is used later in this tutorial and in other tutorials.

In **Tutorial 1: Fundamentals** you set up the **Defect** table. In this chapter, you will set up one additional table:

- DefectArea Defect areas (categories)

This involves some of the same methods that were used to set up the **Defect** table, but using additional features.

You will also use (but not import) the pre-built *Employee* entity from the CodeObjects enterprise component.

Data changes such as these belong with your other data layer metadata, in the **tutorial-model** module.

Working with New Data

This tutorial requires the following steps for the **DefectArea** table, the same steps that you used to set up the **Defect** table in **Tutorial 1**:

1. Import data table definitions for the defect area table (CO_DEFECT_AREA).
2. Define a metadata **entity** for the new data table—entities are used to define higher-level properties of your data, such as validation rules and access rules.
3. Use the entity to define additional characteristics of the data and create relationships between the new data table and the existing defect data table.
4. Define a **data object** based on the entity that will be used to generate the service.

Working with **Employee** data requires a different technique: using a pre-built CodeObjects Platform component module. CodeObjects Platform provides a variety of standard components for assembling applications, including notes, attachments, addresses, and so on. For the Employee data, you will use the pre-built CO_EMP table and Employee entity from the component-model module. The **tutorial-model** module created previously will contain the metadata, and you will use data that is defined in the **CORADS_TUTORIAL** database in SQL Server.

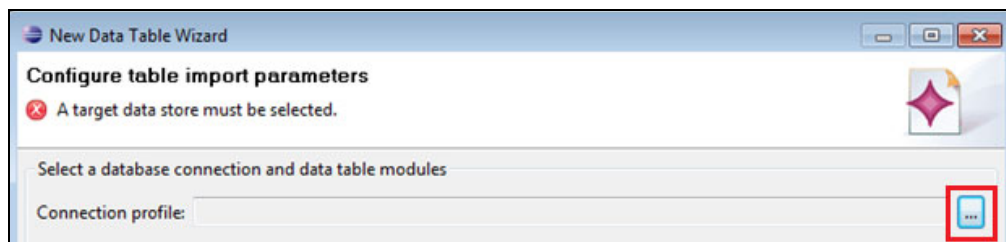
Remember that the high level (structural) metadata entries appear in the **CodeObjects Metadata** pane. When you double-click a metadata node in the Metadata pane, the Editor pane displays all of the lower level metadata in a hierarchical tree, and you can set the node values in the Property pane.

The Procedure

1. Create the DefectArea Data Table by Importing from the Database

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

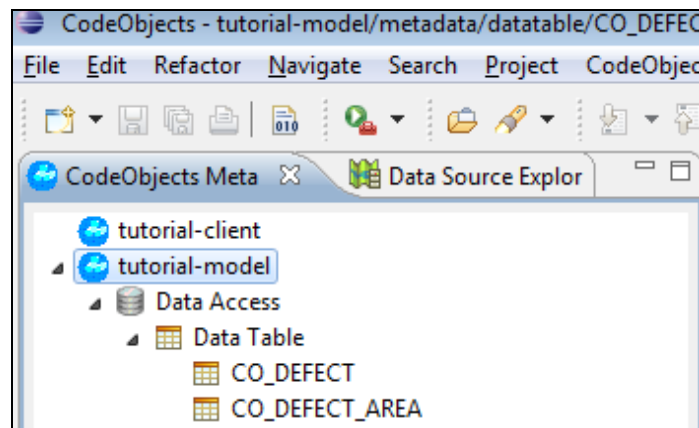
1. In the Metadata pane, right-click the **tutorial-model** module and choose **New → Data Access → Data Table from Database**.
2. On the right end of the **Connection profile** field, click the **Pick** button .



3. Choose the connection profile that you created previously.
 - If your connection profile is active and is already connected to the database, the **Connect** button will be unavailable, so just click **Cancel**.
 - If the **Connect** button is available, click it to connect to the database. Click **OK**.
 - If the connection does not exist, refer to the material in Tutorial 1 starting on page 15 and recreate the connection.
4. On the right end of the **Target datastore** field, click the **Pick** button, choose **AppsDB (application common)**, and click **OK**.

Adding More Data Tables

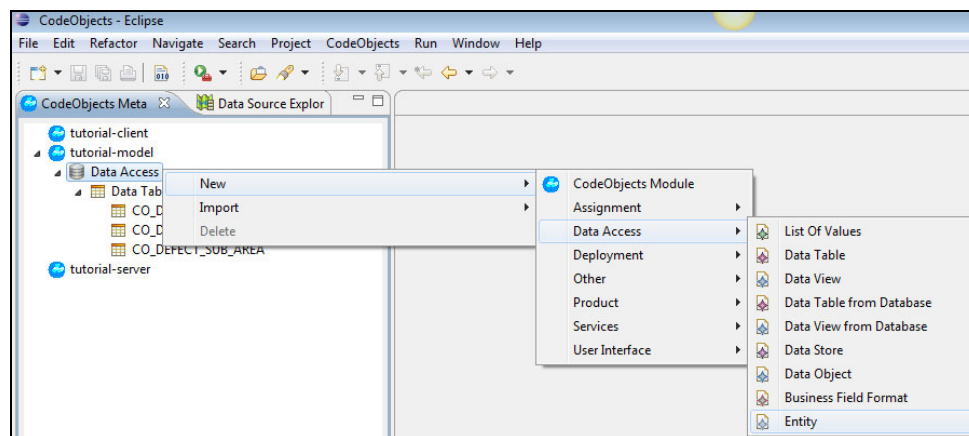
- Click **Select tables to import** to display all the data tables in the database. Choose the table **CO_DEFECT_AREA** and click the **Right Arrow** to list the table in the **Selected** pane.
- Click **OK**, and then click **Finish**. The new imported data table CO_DEFECT_AREA appears in the **CodeObjects Metadata** pane, along with the CO_DEFECT table that you imported in Tutorial 1.



2. Create an Entity for the DefectArea Data Table

Create a metadata *entity* for the **DefectArea** data table that you imported.

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

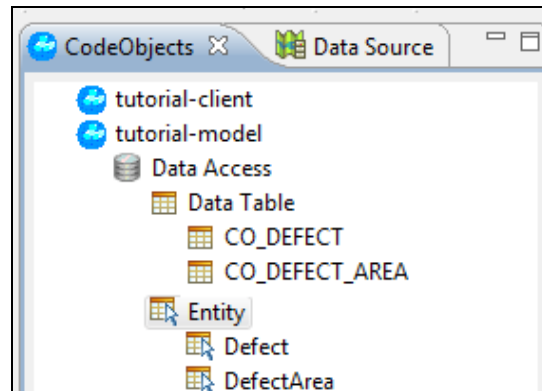


- Define the entity's **Module**, **Name**, **Entity Type**, and associated **Data Table**. Click **Next**.

Module	Name	Entity Type	Data Table
tutorial-model	DefectArea	Simple Entity based on Data Table or Data View	CO_DEFECT_AREA (tutorial-model)

3. Again click **Next** to accept the default values.
4. Click **Finish**.

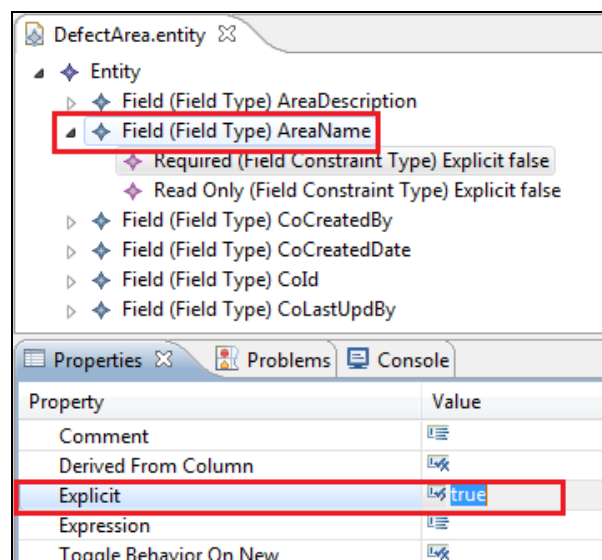
The new entity appears in the metadata tree, below the **CO_Defect** entity that you created in *Tutorial 1*.



3. Create a Required Field in the DefectArea Entity

After defining the defect area entity, it is sensible to make the name of the area a required field.

1. In the Metadata pane, expand the tree through **tutorial-model** → **Data Access** → **Entity** and double-click **Defect Area**.
2. In the Editor pane, expand the node **Field (Field Type) AreaName**, and click **Required (Field Constraint Type) Derived From Column**.
3. In the Property pane, clear **Derived From Column** and set **Explicit** to **true**.



4. Save the changes. Press CTRL+S.

A Note About Data Objects

Recall from **Tutorial 1** in which you created the **Defect** data object that a *data object* consists of one or more data slices.

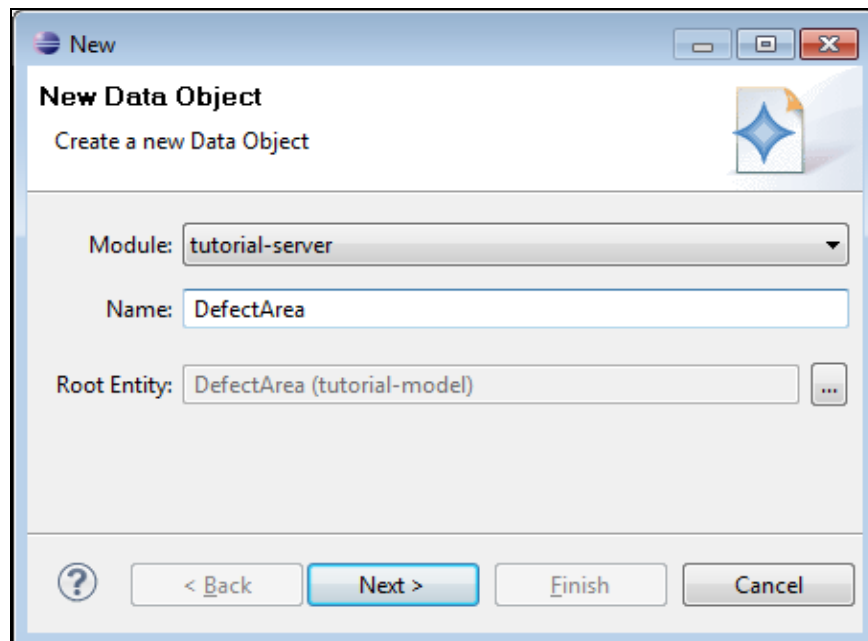
In this section, you will create data objects for *defect areas* and for *employees*.

Although you created previous parts of the data layer in the **tutorial-model** module, you will create the data object in the **tutorial-server** module, which is used to generate services for data access. There are, however, many ways to generate services, and the data object is just one of them.

4. Define a Data Object for DefectArea

To create the **DefectArea** data object:

1. In the Metadata pane, right-click the **tutorial-server** module and choose **New → Data Access → Data Object**.
2. In the dialog box, in the **Name** box type **DefectArea** and for the **Root Entity**, click the **Pick** button, choose **DefectArea (tutorial-model)**, and click **OK**.



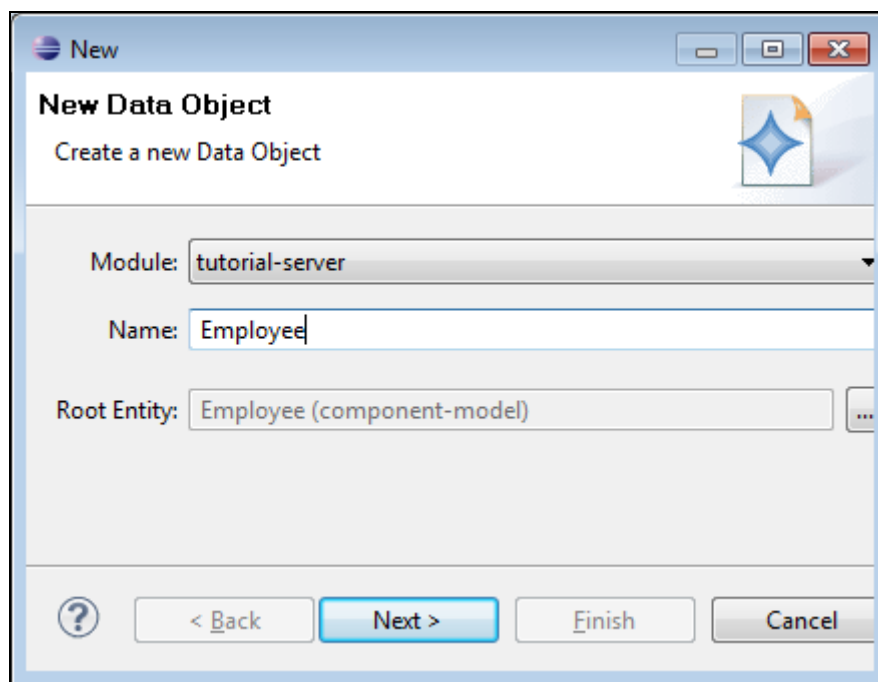
3. Click **Next**.
4. Click within the Configure Data, click **Select All Fields**, and click **Finish**.
5. Save changes to all modules. Click **File → Save All**.

The DefectArea object appears in the Metadata pane.

5. Create a New Data Object for the Employee Records

Because the defect tracking system will assign defects to employee, it requires a place in the application to manage employee records. You will use the prebuilt Employee entity from the component-model module. To create the metadata for the employee component model:

1. Right-click **tutorial-server** and choose **New** → **Data Access** → **Data Object**.
2. Set **Name** to **Employee**.
3. For the **Root Entity**, choose **Employee (component-model)**.



4. Click **Next**.
5. From the **Root (component-model.Employee)** list, in addition to the pre-selected fields select the check boxes for the following fields. These fields will appear in the tutorial application for the user to administer.
 - **Department**
 - **EmailAddress**
 - **EmpStatus**
 - **Extension**
 - **FirstName**

- **JobTitle**
- **LastName**

6. At the bottom of the list, select the child node **LoginUsers (component-model.LoginUser)**. This component is used to record login usernames and passwords.
7. Click **Finish**.

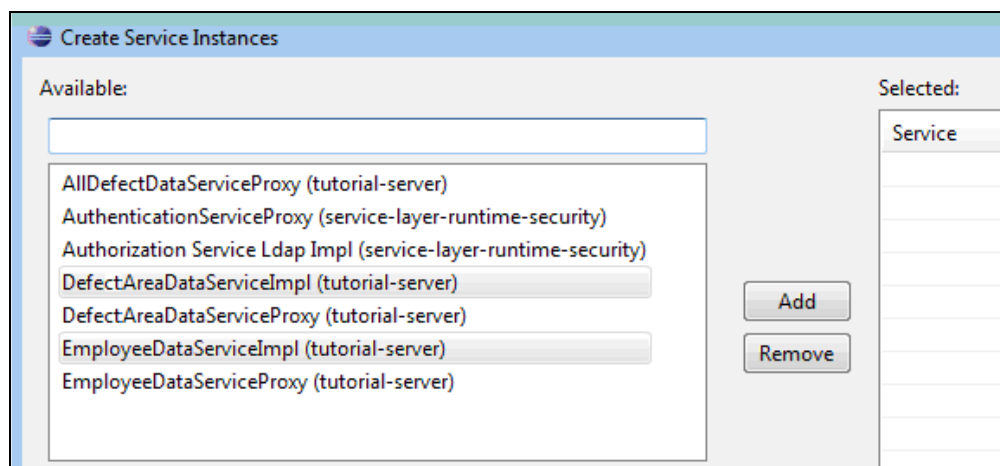
Note that this process does not create data table or data entity metadata entries. The data object itself is linked to the pre-built data table(s).

6. Add Data Objects to the System Metadata

In order to access the data object in the application, add them to the **System** metadata to deploy together.

1. In the Metadata pane under **tutorial-server** → **Deployment** → **System**, double-click **Tutorial** to open the system node (**Tutorial.system**).
2. In the Editor pane, click the **Add** button.
3. In the **Create Service Instances** dialog box, under **Available**, select the following services, and then click the **Add** button to list them under **Selected**.

- **DefectAreaDataServiceImpl (tutorial-server)**
[generated from the DefectArea data object created above]
- **EmployeeDataServiceImpl (tutorial-server)**
[generated from the Employee data object created above]



4. Click **OK**.
5. Save your changes. Press CTRL+S.

Defining Relationships Between Entities

In this chapter, you will establish relationships among your between—logical connections between data that you specify as metadata.

Overview of Entity Relationships

Entities are the heart of data management in CodeObjects Platform. Data *tables* map the physical data and data *objects* are used to generate services.

Entities may have relationships among them, most commonly a parent-child relation. In order to define a relationship between entities, CodeObjects requires:

- **A link key** Link keys are simply metadata (attached to an entity), which indicate fields that can be linked to another entity. In relational database terminology, a link key is like a foreign key. In this example, the **DefectAreaId** field in the **Defect** entity contains the key of the associated **DefectArea** record. (In the **DefectArea** entity, the record key is stored in the **Cold** field.)

Note that if the entity has a unique key, that can also serve as a link key.

A link key consists of the foreign key field, plus a name for the link key.

- **An association to another table** Associations are one-way connections from a source to a destination. For every association, there must be a source key, a source entity, and a destination key and entity.

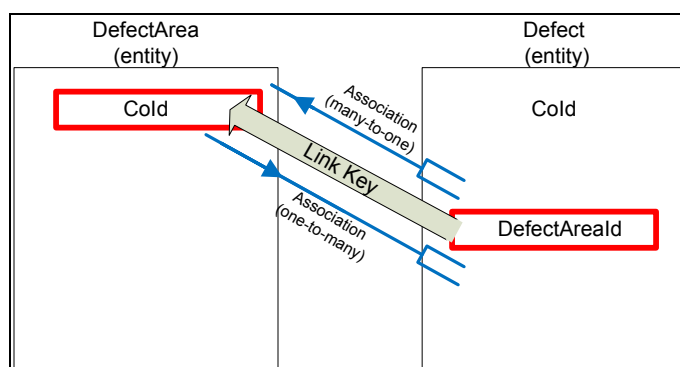
You define the association metadata for an entity, and that entity is the source entity.

Association metadata properties are:

- **Source key** Link key or field in the source entity
- **Destination key** Link key or field in the destination entity
- **Associated Entity** Destination entity for the association

- Type of association** Associations between entities can be one-to-many (or parent-to-child) or many-to-one (or child-to-parent). In this example, one defect area can have multiple defects (one-to-many). Thus in this example, there is a one-to-many association from **DefectArea** to **Defect**. At the same time, many defects can belong to the same defect area, so this is a many-to-one association from **Defect** to **DefectArea**.

The following diagram shows how you will associate the two entities. You'll create a **link key** in order to associate the **DefectAreaId** foreign key field in the **Defect** entity with the **Cold** field in the **DefectArea** entity. Then, using the link key, you'll create two associations that will enable you to access/view the defect area record while working with defects *and* display/access the defect records associated with a given defect area. If you didn't need to access in both directions, you wouldn't need both associations.



So, to define the connections between the **Defect** and **DefectArea** entities, you create *link key* metadata in the **Defect** entity to link the **DefectAreaId** foreign key field to the key field of the **DefectArea** entity.

- Define *association* metadata for the **Defect** entity with the following properties:
 - Many-to-one.
 - Source entity is **Defect** (where you create the association).
 - Source key is the link key that links the **DefectAreaId** field in the **Defect** entity to the **Cold** field of the **DefectArea** entity.
 - Destination entity is **DefectArea**.
 - Destination key is the **Cold** field in the **DefectArea** entity.

- Define *association* metadata for the **DefectArea** entity with the following properties:
 - One-to-many.
 - Source entity is **DefectArea** (where you create the association) .
 - Source key is the **Cold** field in the **DefectArea** entity.
 - Destination entity is **Defect**.
 - Destination key is the link key that links the **DefectAreaId** field in the **Defect** entity to the **Cold** field of the **DefectArea** entity.

Note that *after a link key is defined, it can be used just like a field to create multiple associations*. The link key **DefectAreaId** that you create will be used in both of the associations that you create.

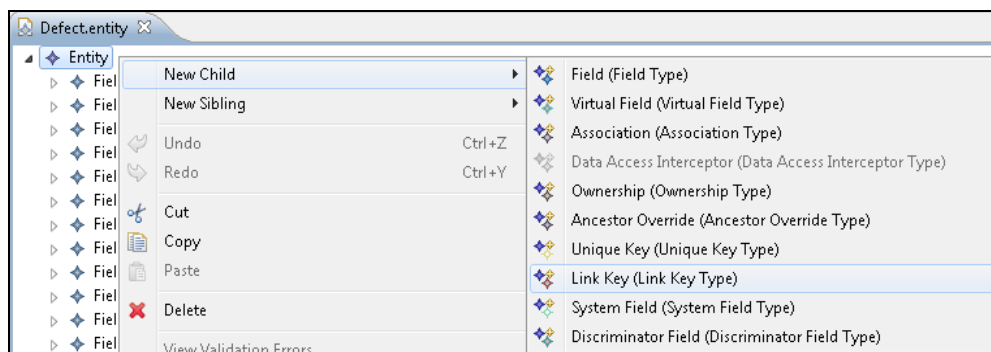
Note also that *you may not need to define the relationship in both directions*—you may only need one direction depending on your scenario. In this case, for example, there may be no need to define the one-to-many relationship between defect area and defect. However, you have defined the link in both directions in this tutorial to demonstrate how it works.

The Procedure

1. Define the Link Key to DefectArea in the Defect Entity

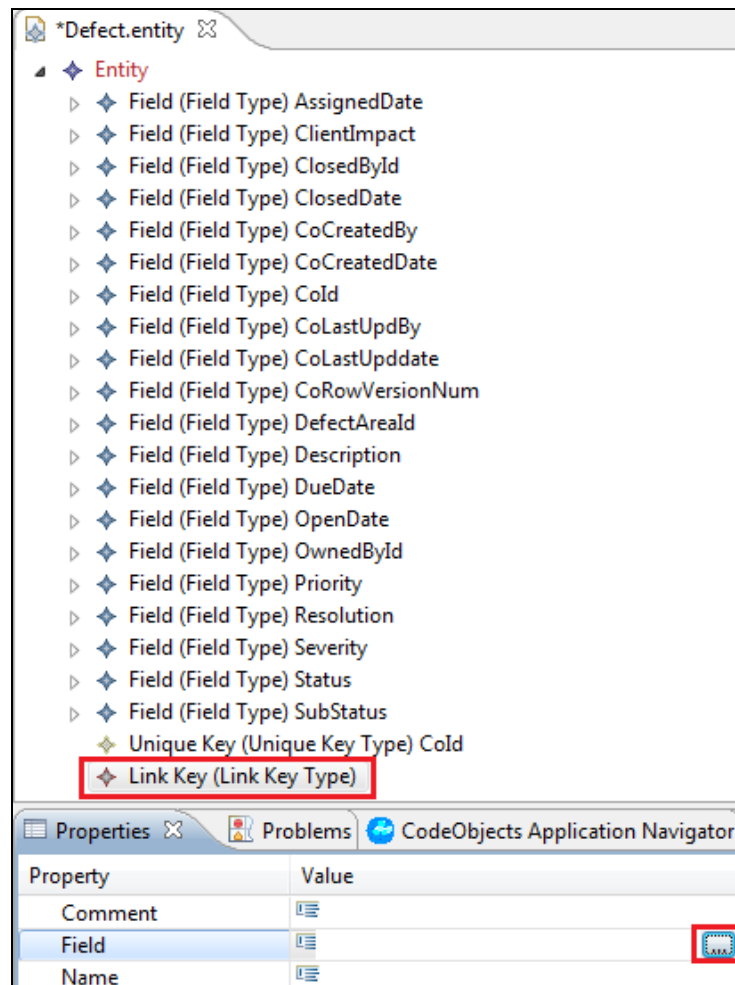
You must define the link key between the foreign key column in the **Defect** entity and the **DefectArea** entity.

1. In the metadata pane under **tutorial-model** → **Entity**, double-click the **Defect** entity to display the entity's properties in the Editor pane.
2. Right-click the top node **Entity** and choose **New Child** → **Link Key (Link Key Type)**.

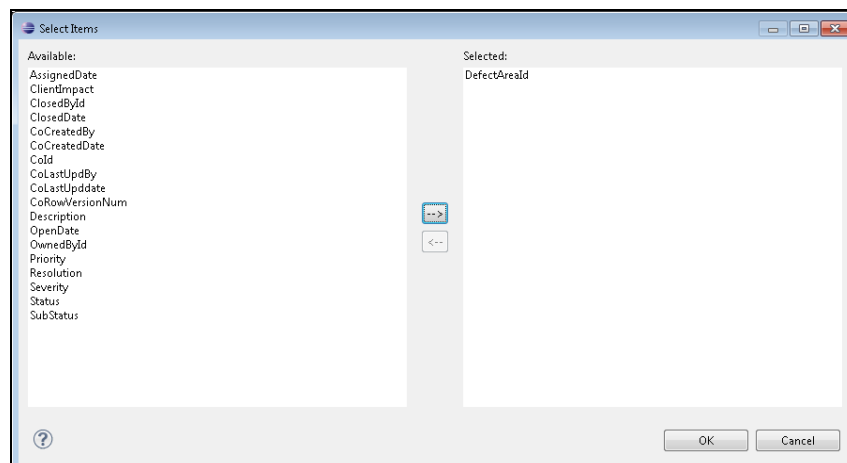


Defining Relationships Between Entities

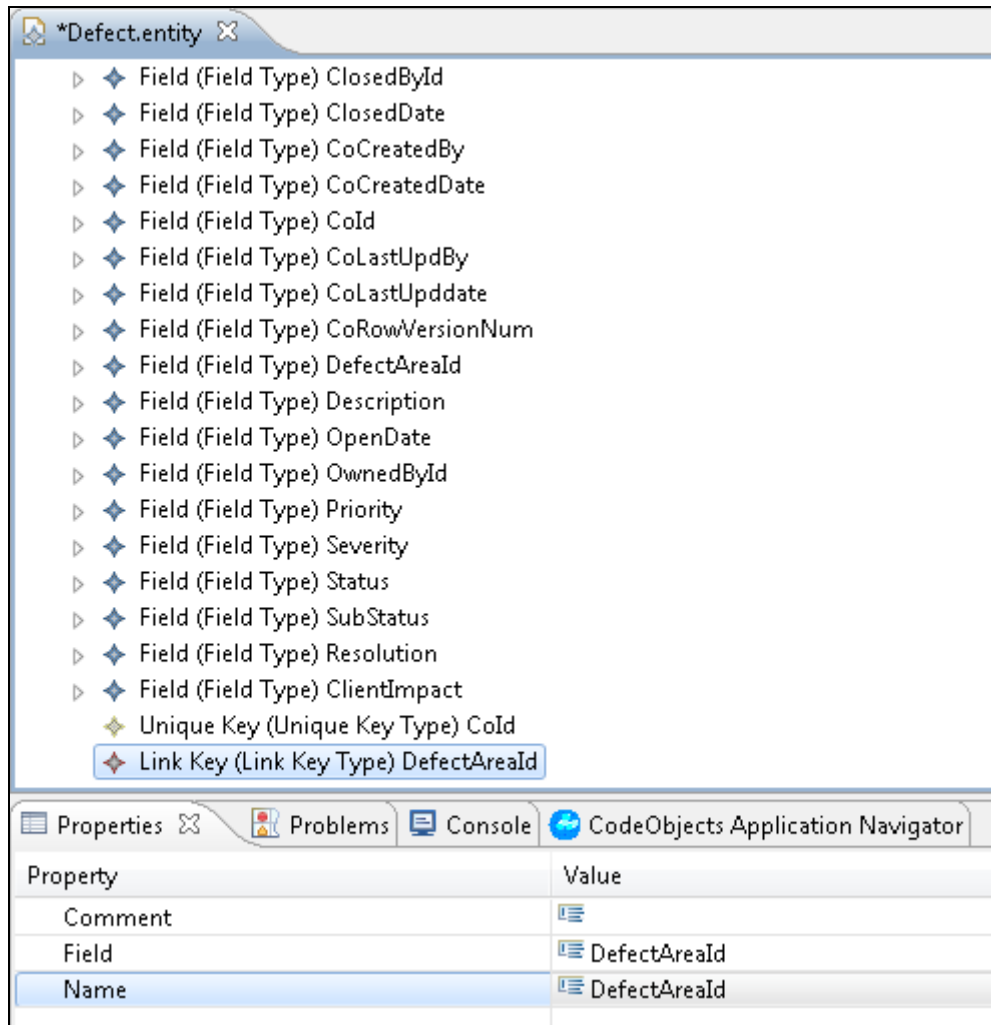
3. In the Property pane, for **Field** click the **Pick Property** button .



4. In the **Select Items** dialog box, under **Available**, click **DefectAreaId**, click the **Right-Arrow** button to place the item under **Selected**, and click **OK**.



- For the **Name** property, enter **DefectAreaId** (the same as the foreign key column).



- Save your changes. Press CTRL+S.

TIP

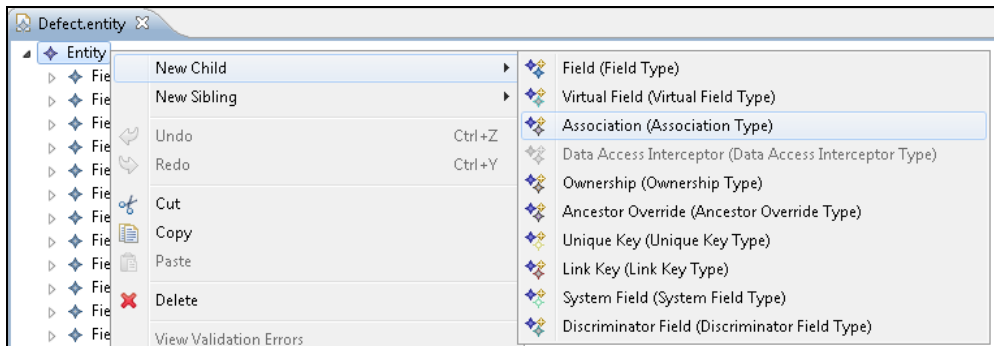
Some of the items that you create in the CodeObjects Workbench are available immediately whether or not you save your file. For example, you established a database connection, and then used it to import data, but you didn't save your file. However, properties must be saved before they can be used. Link keys and their properties will not be available to other entities unless they have been saved.

Defining Relationships Between Entities

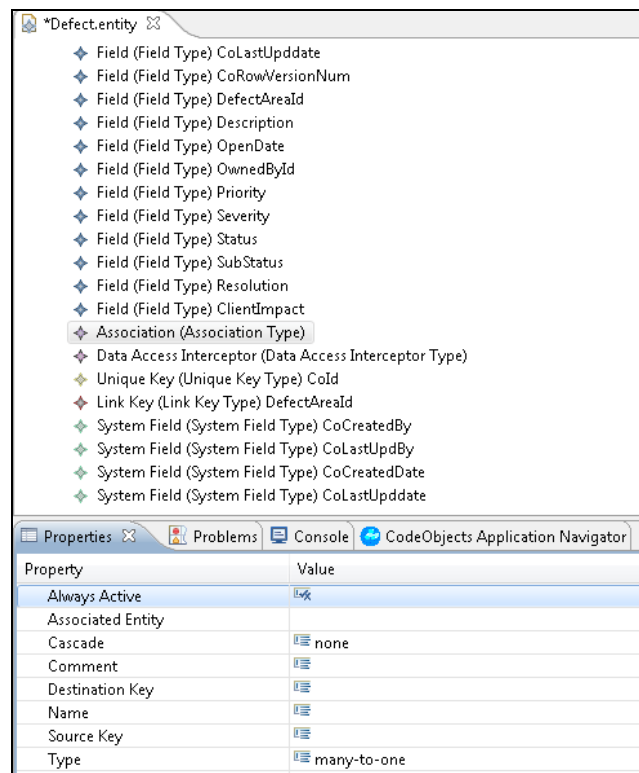
2. Define the Association Between the Defect Entity and the DefectArea Entity

Define the many-to-one association between the Defect entity and the DefectArea entity; that is, many defects can be assigned to a single defect area.

1. In the Editor pane under the **Defect.entity** tab, right-click the top node **Entity** and click **New Child** → **Association (Association Type)**. The association is created.

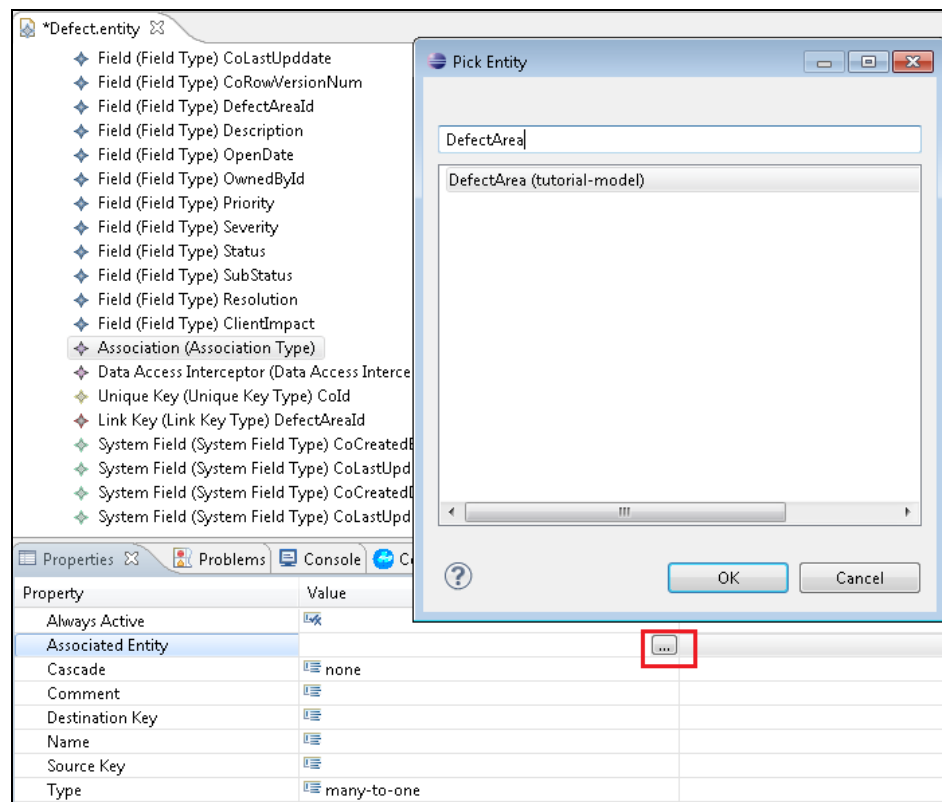


2. In the Property pane, click **Associated Entity**, and then click its **Pick Property** button. Choose **DefectArea (tutorial model)** and click **OK**.



TIP

To easily find an entity in a long **Pick Entity** list, type the first word of the entity's name in the text box. Only entities that match the name will appear in the list. For some lists, the first letters you type immediately start to filter the list. In the previous step above, typing just the letter **d** reduces the list from several hundred entities to ten. The typed letters **de** are sufficient to offer an easy selection from a mere half dozen entities.

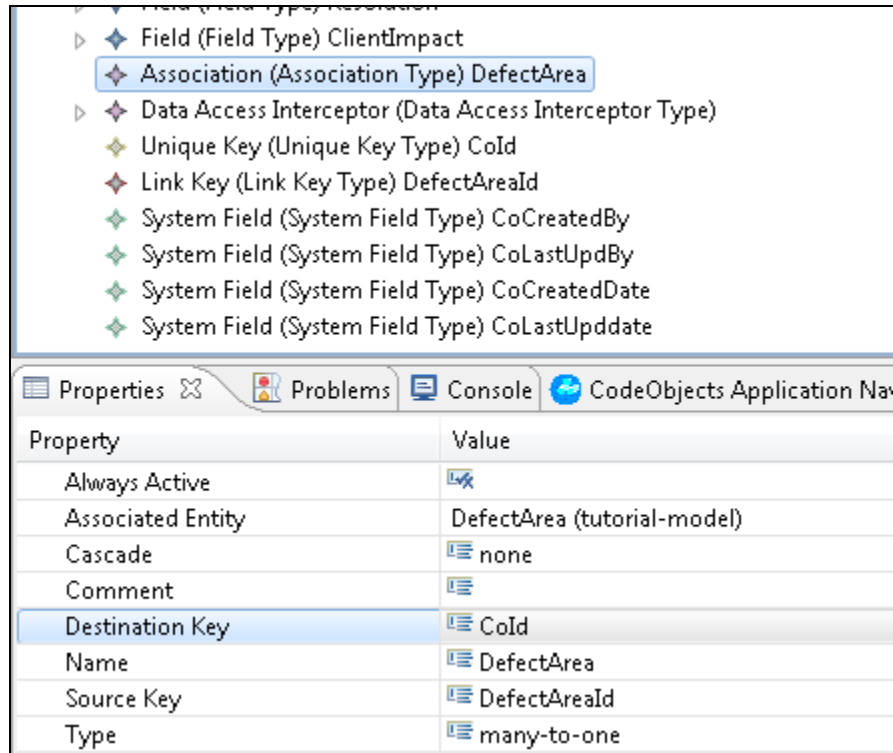


3. Set the other properties.

Property	Value
Associated Entity	DefectArea (tutorial model)
Cascade	none
Destination Key	Cold
Name	DefectArea
Source Key	DefectAreaId
Type	many-to-one

4. Save changes to all modules. Click **File** → **Save All**.

The final values in the Properties pane should look like the following:



TIP

When creating an association, always set the **Type** to match the desired association type *first* so that CORADS can preset the appropriate defaults. In this example, the type of association is already **many-to-one**. When creating a one-to-many association, change the **Type** to be **one-to-many** first.

3. Define the One-to-Many Association from the DefectArea Entity to the Defect Entity

In the previous section, you defined a many-to-one association from **Defect** to **DefectArea**. In this section, you'll create a one-to-many association. One defect area can have many different defects. This is a standard one-to-many relationship.

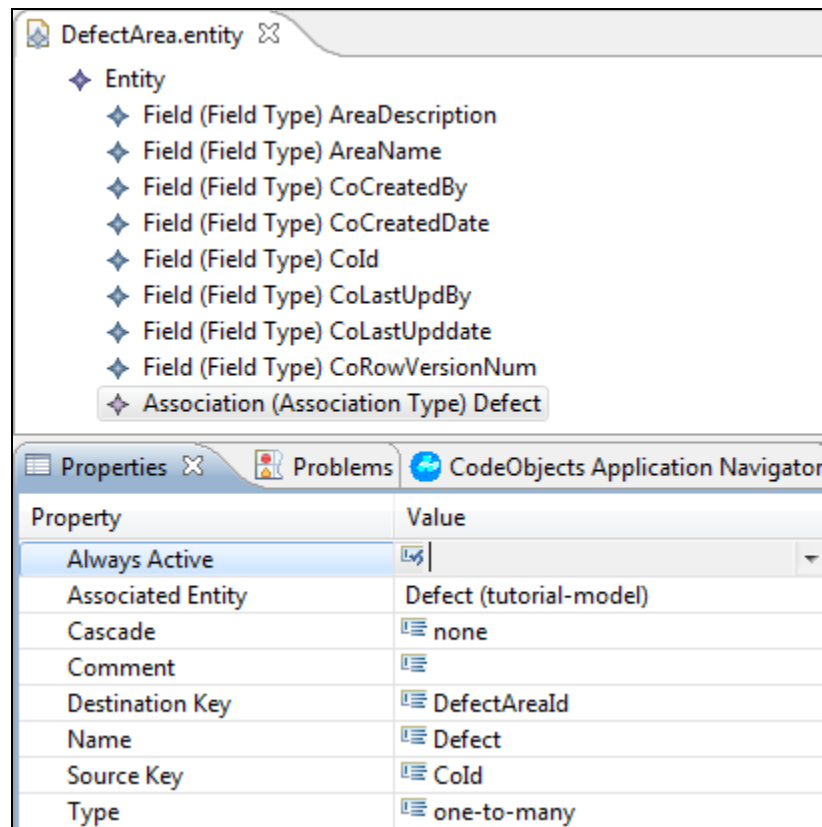
Create the one-to-many association from Defect Area to Defect.

5. In the Editor pane under the **DefectArea.entity** tab, right-click The **Entity** node, and then click **New Child** → **Association (AssociationType)**.

Defining Relationships Between Entities

6. In the Property pane, change the **Type** of the association to **one-to-many**, and then set the other properties.

Property	Value
Associated Entity	Defect (tutorial model)
Cascade	none
Destination Key	DefectAreaId
Name	Defect
Source Key	Cold
Type	one-to-many



7. Save the association. Press CTRL+S.

Creating New Widgets and Views

With two new data objects defined (DefectArea and Employee created earlier in this tutorial), you must be able to work with that data in the user interface—in this case, to perform standard CRUD (Create/Retrieve/Update/Delete) operations on them. In this chapter, you will set up a new widget that contains two views: one view for Defect Area records and one view for Employee records.

Review of Widgets and Views

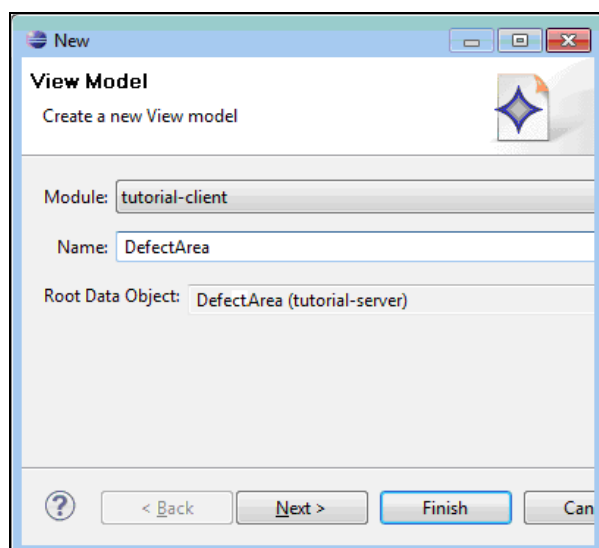
Recall that a widget appears as a tab within the client interface. You will create a new administration tab (widget), which will have two views, one for Defect Areas and one for Employees.

The Procedure

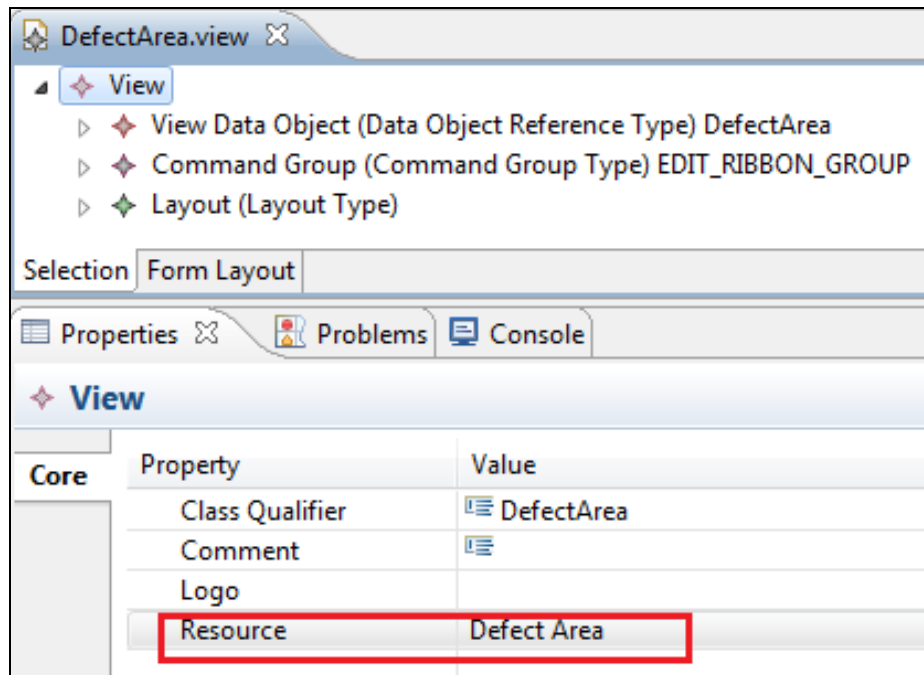
1. Create the View for Defect Areas

The new **DefectArea** data object requires a view in order to appear in the client interface. Create the required view:

1. Right-click **tutorial-client** and choose **New** → **User Interface** → **View**. Set the **Name** of the view to **DefectArea** and the **Root Data Object** to be **DefectArea (tutorial-server)**.

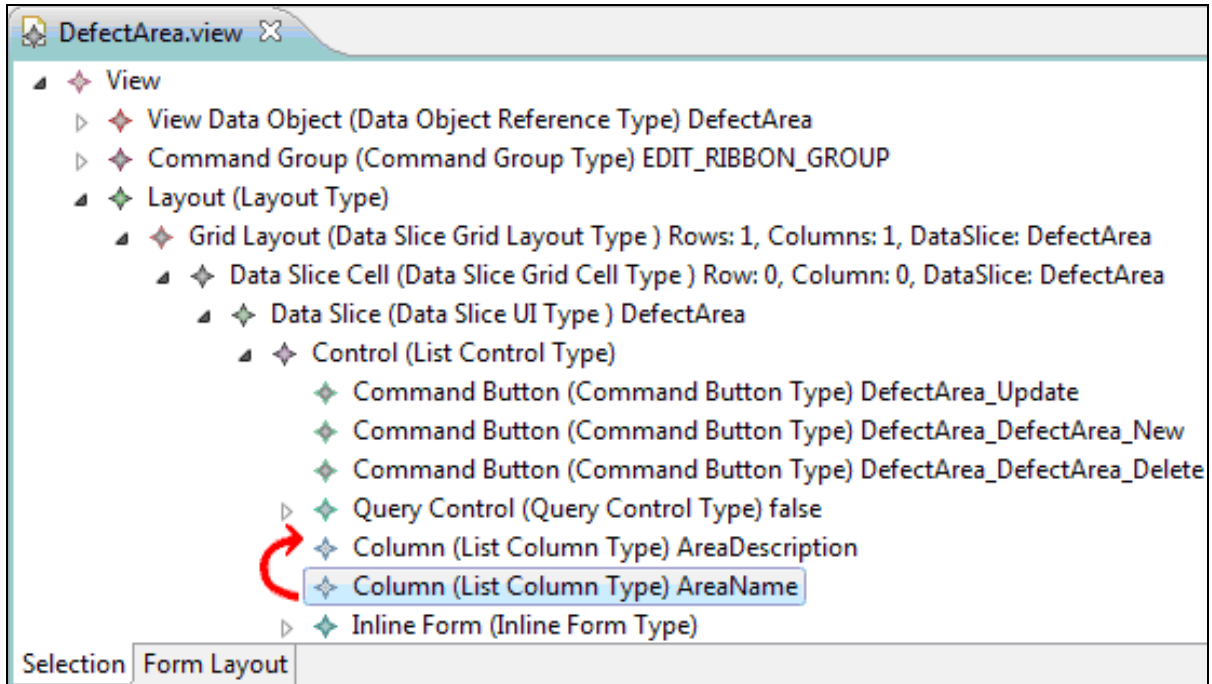


2. Click **Next**. Review the settings, but leave them at their default values and click **Finish**. (Do *not* just click **Finish** without clicking **Next** first, because it assigns some values behind the scenes.)
3. In the Editor pane under the **DefectArea.view** tab, click the **View** node.
4. In the Property pane, change the text string that will be displayed on the client interface (resource string).
 - a. Click **Resource**, and then click the **Pick Property** button.
 - b. In the **Pick a Resource** dialog box, in the text box, type **Def***, and then click **Search**.
 - c. In the list pane, click **Defect Area**.
 - d. Click the **Pick Selected Resource** button.



5. In the Editor pane, click the node **View Data Object (Data Object Reference Type) Defect Area**.
6. In the Property pane, set **Run Query Command On Load** to **true**.
7. In the Editor pane, expand the tree for the **Layout (Layout Type)** node through **Control (List Control Type)**.

8. Drag the node **Column (List Column Type) AreaName** above **Column (List Column Type) AreaDescription**.

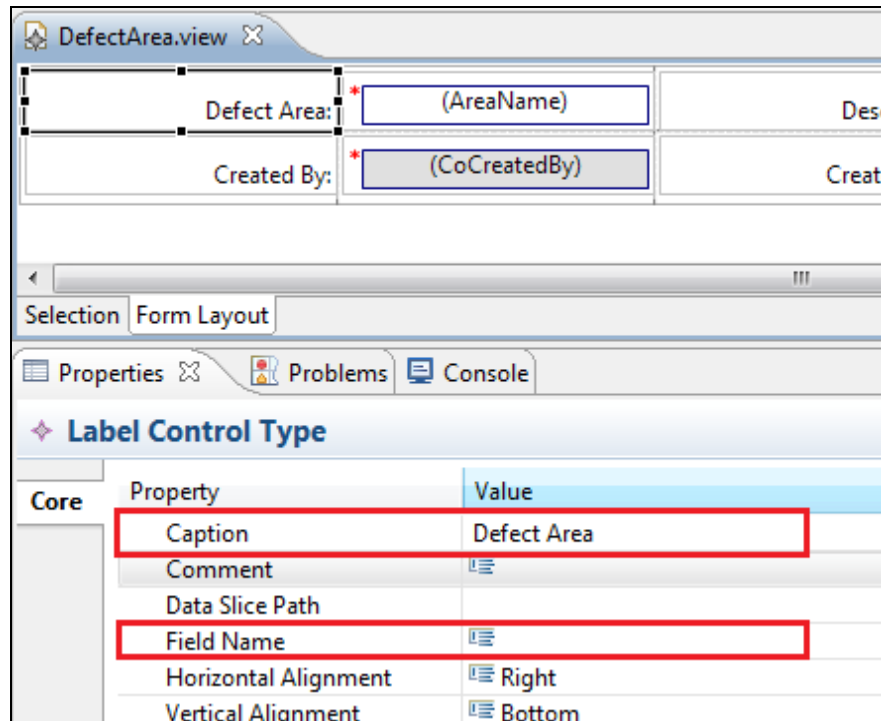


9. Right-click the bottom node **Inline Form (Inline Form Type)** and choose **Open Layout Editor** to switch to the Form Layout Editor.
10. Click to the right of the **AreaName** field, and in the Property pane, set the **Number of Rows** to **2**. Press CTRL+S to save the changes and update the layout grid.
11. Change the data entry form grid to look like the following image. To set the contents of a label cell, click within the cell and edit the contents in the Property pane.

Defect Area:	(AreaName)	Description:	(AreaDescription)
Created By:	(CoCreatedBy)	Created Date:	(CoCreatedDate) [Calendar Icon]

- a. Rearrange the **AreaName** and **AreaDescription** labels and fields, and then drag the right edge of the **AreaDescription** field so that it extends through three cells.

- b. For the **AreaName** label, set the **Caption** to **Defect Area** and clear the **Value** for the **Field Name**.



TIP

In the **Pick a Resource** dialog box, type the first several letters and an asterisk wildcard (*), and then click **Search** to filter the available entries in the list pane. For example, type **Def *** and click **Search** to locate **Defect Area** easily.

- c. For the **AreaDescription** label, set the **Caption** to **Description**, and clear the **Value** for the **Field Name**.

TIP

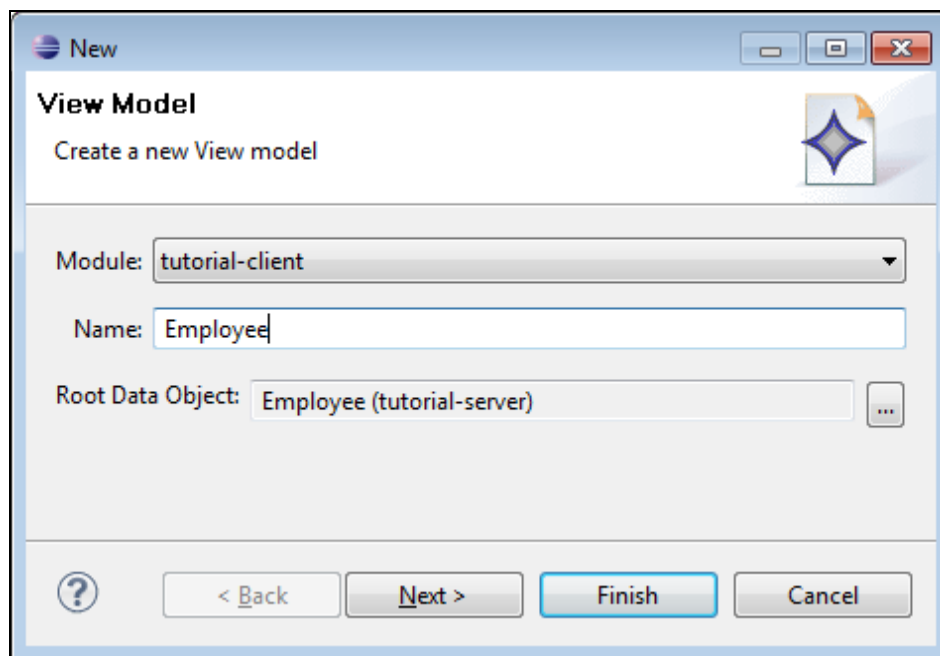
When you click a **Value** field, and on the right edge of the field it provides a list button , you need not bother to click precisely on the button. Just click again anywhere within that Value field to open the list. If your mouse has a scroll wheel, you can scroll to see the available items in the list instead of dragging the slider bar.

- d. In the form Palette, under **Label & Field (H)**, click the palette entry, and then click the appropriate layout cells to add **CoCreatedBy** and **CoCreatedDate**.
- e. Set the **Caption** labels for the two new entries to **Created By** and **Created Date** respectively, and then clear their **Field Name** values.

NOTE

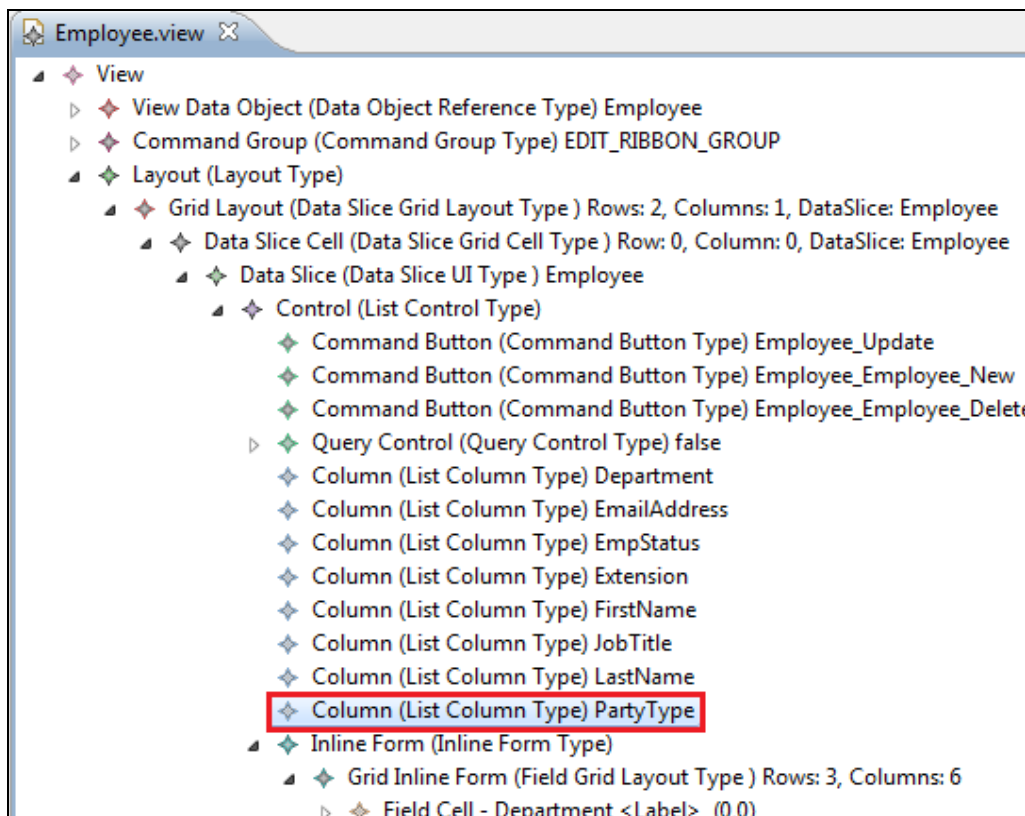
In **Tutorial 2**, you set the **Resource** property for fields in the Entity metadata. Properties that are set in the entity metadata are enforced by every module that accesses that metadata, and any resource strings that you created will automatically appear on the form. With the technique that you are using here, setting the caption property in the form, metadata is local to this form only.

2. Create the View for Employee Data
 1. In the Metadata pane, right-click **tutorial-client**, and then choose **New** → **User Interface** → **View**. Set the name as **Employee** and as the **Root Data Object** choose **Employee (tutorial-server)**.



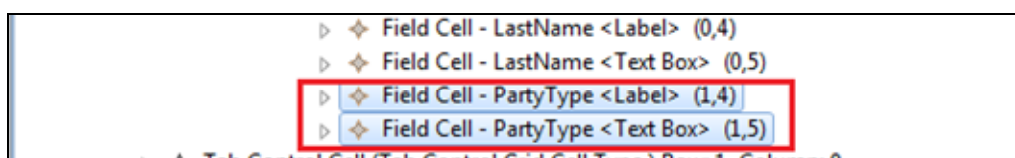
2. Click **Next**. Review the settings, but leave them at their default values and click **Finish**.
3. Click the node **View Data Object (Data Object Reference Type) Employee** and set **Run Query Command On Load** to **true**.

4. A Data Object sub-node contains the **PartyType** field, which is internally managed by CodeObjects Platform and is not appropriate to show to the end user. Delete it from the view so that users won't see it:
 - a. Expand the tree from the node **Layout (Layout Type)** (via the sub-node **Data Slice**) through the node **Control (List Control Type)**.
 - b. Near the bottom of the branch, click **Column (List Column Type) PartyType**, and then press the DELETE key to delete it.

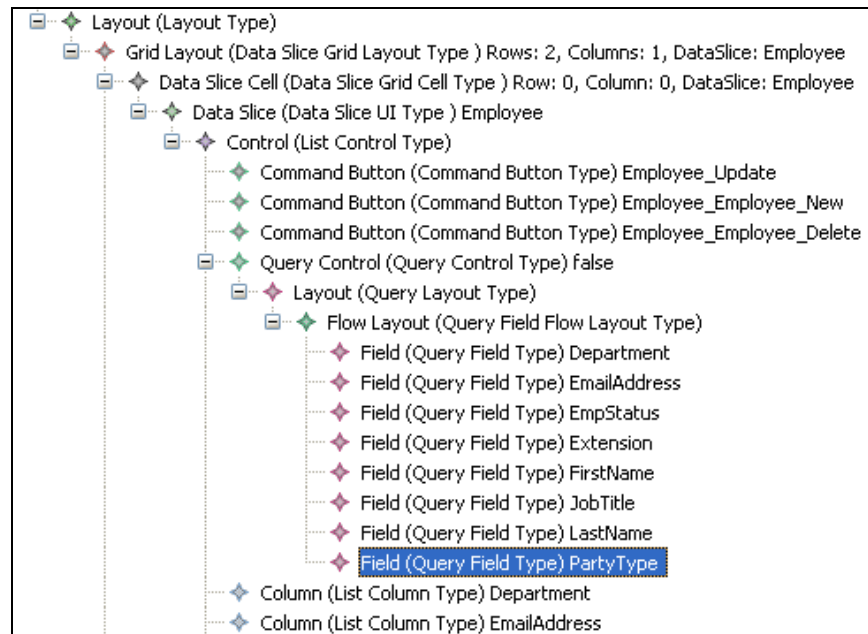


5. Expand the tree from the node **Inline Form (Inline Form Type)** two levels to reveal (among many others) two **Field Cell – PartyType** nodes.
6. Click and delete them, too.

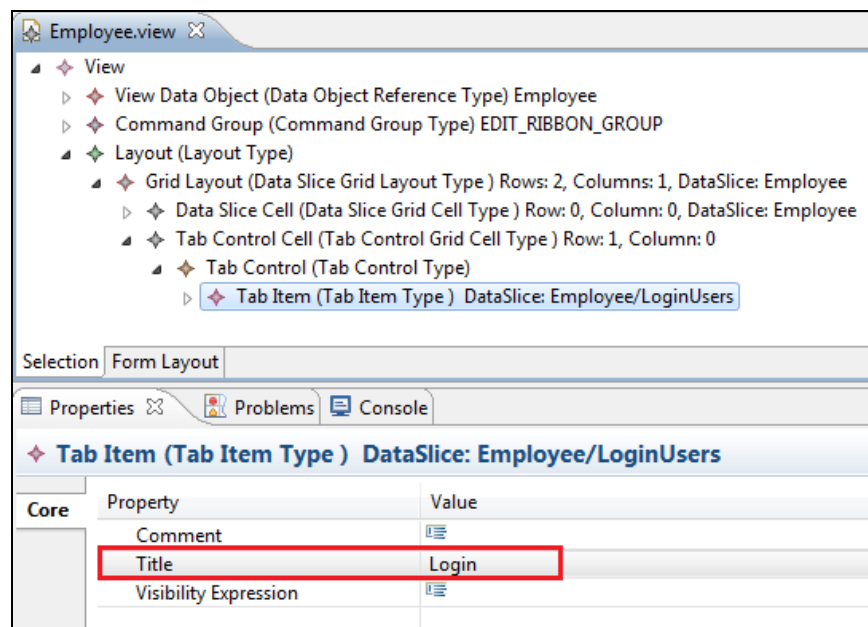
Alternatively, you can right-click **Inline Form (Inline Form Type)** and choose **Open Layout Editor** to see them in the form layout and delete them there. (To return to the tree view, at the bottom of the Editor pane, click the **Selection** tab.)



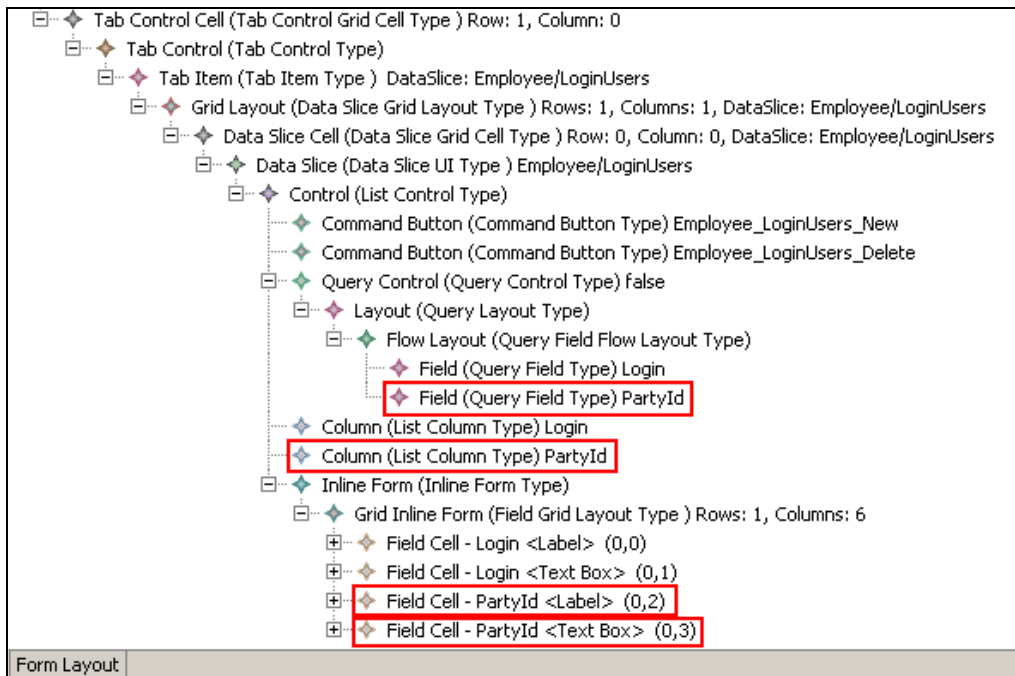
7. Fully expand the tree from the node **Query Control (Query Control Type) false**, and delete the node **Field (Query Field Type) PartyType**.



8. Expand the tree from the node **Tab Control Cell (Tab Control Grid Cell Type) two** levels.
9. Click the node **Tab Item (Tab Item Type) DataSlice: Employee/LoginUsers**, and then in the Property pane, change the **Title** to **Login**.



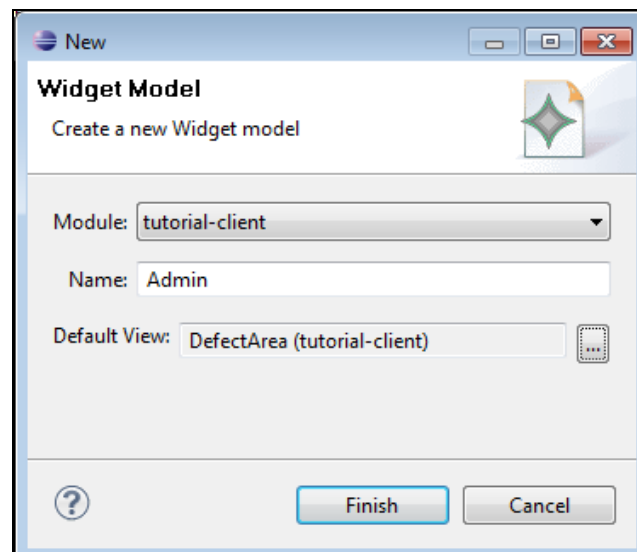
10. Fully expand the tree further all the way through **Grid Inline Form (Field Grid Layout Type)** and delete the four nodes that contain **PartyId**.



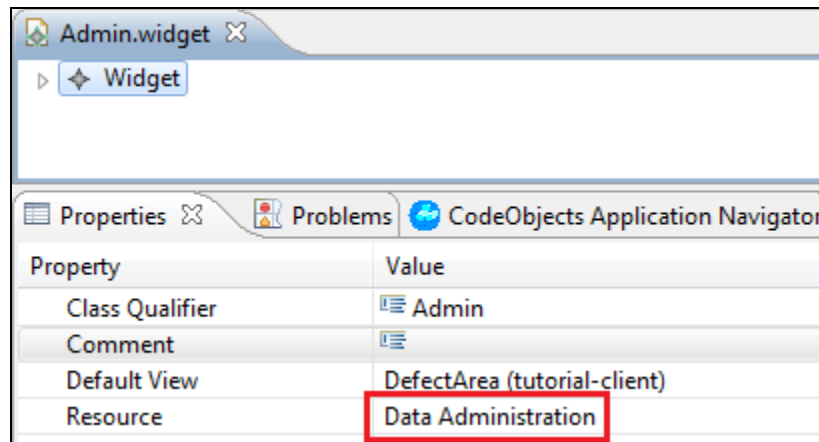
3. Create a New Widget

Widgets are containers for views. Because the two views added to the application have been defined, create the widget that will display those views in the client interface.

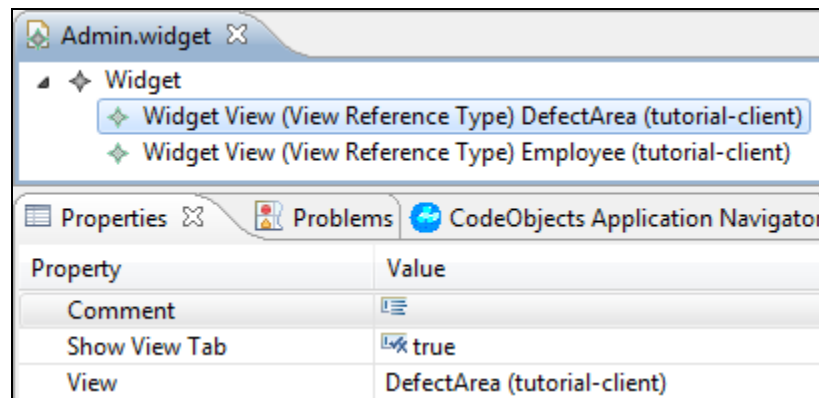
1. In the Metadata pane right-click **tutorial-client** and choose **New** → **User Interface** → **Widget**. Set the **Name** of the widget to **Admin** and the **Default View** to **DefectArea (tutorial-client)**.



2. In the Editor pane, click the node **Widget**, and then in the Property pane, set **Resource** to **Data Administration**.



3. In the Editor pane, right-click **Widget** and choose **New Child** → **Widget View (View Reference Type)**. Click the new **Widget View (View Reference Type)** node, and then in the Property pane set **View** to **DefectArea (tutorial-client)**.
4. In the Editor pane, right-click **Widget** again and in the same way, add an **Employee (tutorial-client)** view to the widget.

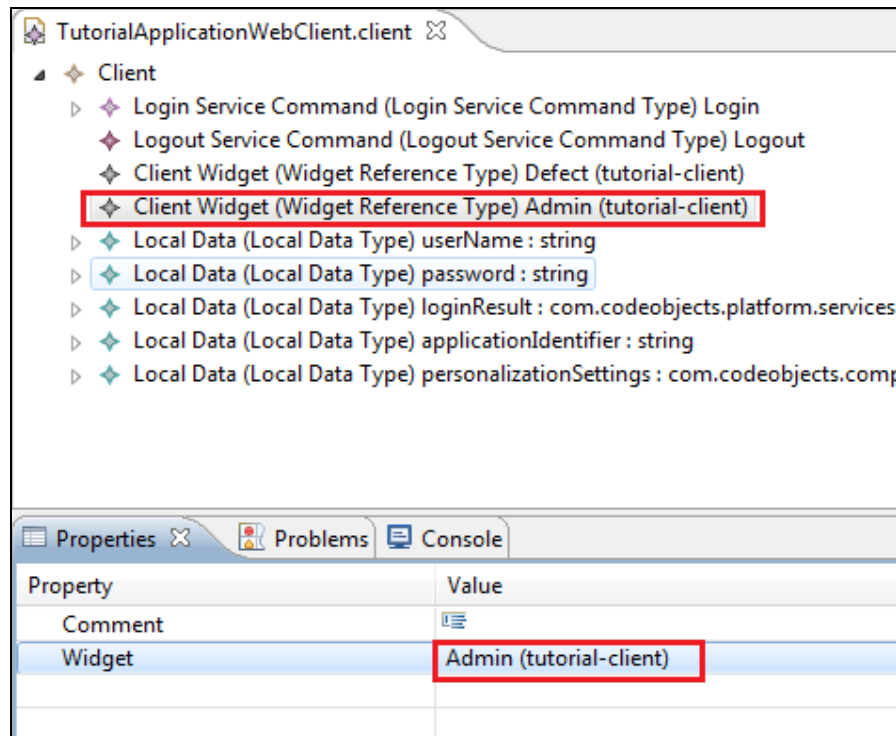


4. Add the New Widget to the Client

Recall that the high level **Client** metadata is used to build the client and is stored in the **tutorial-client** module. To include your new widget in the client, you must update the Client metadata entry to specify that widget.

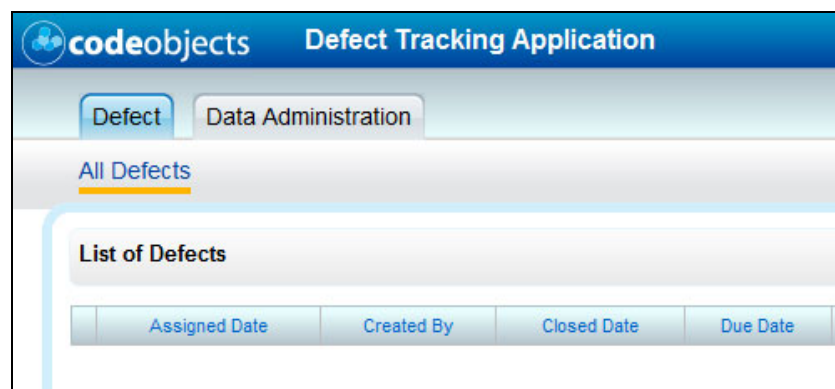
1. In the Metadata pane, expand **tutorial-client** → **Deployment** → **Client** and double-click **TutorialApplicationWebClient** to open **TutorialApplicationWebClient.client**.

2. In the Editor pane, right-click the node **Client** and choose **New Child** → **Client Widget (Widget Reference Type)**.
3. In the Property pane, set **Widget** to **Admin (tutorial-client)**.



Test the Result

1. Ensure that all changes are saved. Click **File** → **Save All**.
2. Archive and re-deploy the client. Notice the new tab (widget) **Data Administration**. If you added some records in previous tests of the application, those records appear automatically without having to click the **Search** button.



Creating New Widgets and Views

3. Click the **Data Administration** tab and use the **Defect Area** view to add some defect categories. (You will use the defect areas later.)

The screenshot shows the 'codeobjects Defect Tracking Application' interface. The 'Data Administration' tab is active, and the 'Defect Area' view is selected. The view displays a table with three rows: 'Area 1', 'Area 2', and 'Area 3', each with a corresponding 'Defect Area' description. Below the table, there are input fields for 'Defect Area' (set to 'Area 3'), 'Description' (set to 'Defect Area 3'), 'Created By' (set to 'couser'), and 'Created Date' (set to '3/11/2011 4:55 PM').

	AreaName	AreaDescription
▼	Area 1	Defect Area 1
▼	Area 2	Defect Area 2
▲	Area 3	Defect Area 3

Defect Area: Description:
Created By: Created Date:

4. Click the **Employee** view and add a few employees. (You will use the employee records later.)

The screenshot shows the 'codeobjects Defect Tracking Application' interface. The 'Data Administration' tab is active, and the 'Employee' view is selected. The view displays a table with four rows of employee data. Below the table, there is a 'Login' section with a table showing a single login record for 'swombat'.

	Department	Email Address	Status	Extension	First Name	Title	Last Name
▼	Engineering		Active	123	Scott	Employee	Wombat
▼	Engineering		Active	124	Marissa	Manager	Koala
▼	Engineering		Active	125	Dianne	Employee	Emu
▼	Engineering		Active	126	Peter	Employee	Opal

Login

	Login
▼	swombat

Setting up Virtual Fields and Dynamic Lists Of Values (DLOVs)

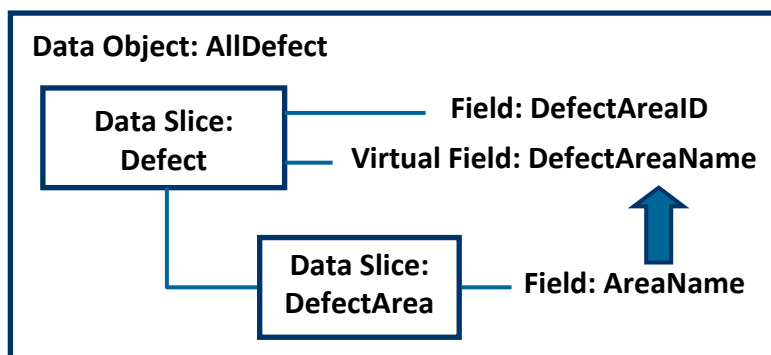
This chapter introduces you to virtual fields for creating a simple virtual field in the application. You will also set up a dynamic list of values, a list that has values drawn from a data table rather than from a static list defined in metadata.

Virtual Fields

CodeObjects platform enables you to define arbitrary *virtual fields* that are computed fields, derived from other fields in different data slices. A virtual field is a placeholder that can be used by other users, programming extensions, and so on.

In the application, the **Defect** table has a **DefectAreaId** column, which is a foreign key to the CO_DEFECT_AREA table. You need a drop-down list to show the end user the list of defect areas, rather than the IDs—displaying a Defect Area field instead of an ID field. However, in the table schema, you only have a DefectAreaId field, so this provides a perfect case for creating a virtual field for DefectArea.

The virtual field you will create is in the **Defect** data slice, and it will derive its value from **DefectArea**. Conceptually, you should show the text value of the defect area from the record to which **DefectAreaId** points. So, you must bring the value of the field **AreaName** in **DefectArea** to the **Defect** data slice. To achieve this, you will create a new virtual field with an expression that describes how to populate it.



Dynamic Lists

In **Tutorial 2**, you defined drop-down lists by typing the lists into the metadata. These static lists won't work when data values must be drawn from another database—for that you need a **dynamic list**. You'll refer to dynamic lists or dynamic lists of values (DLOVs).

The Procedure

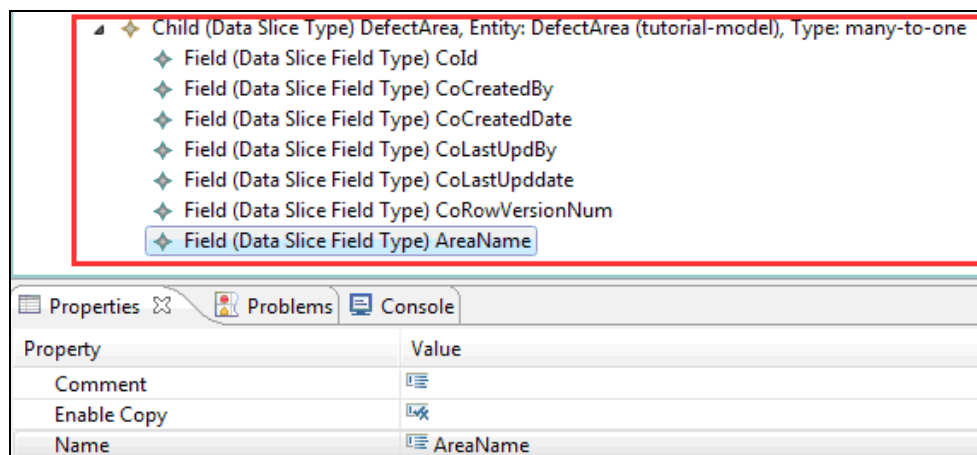
1. Add a Data Slice to the Data Object

You will add a new data slice **DefectArea** to the **AllDefect** data object. This new data slice provides a way to retrieve the defect area value for your virtual field.

1. In the Metadata pane, under **tutorial-server** → **Data Access** → **Data Object**, double-click **AllDefect** to display the data object in the Editor pane.
2. Right-click the node **Root (Data Slice Type)** and choose **New Child** → **Child (Data Slice Type)**.
3. Set the following values. Note that the property **Paging Size** is set to 1 because one defect can have one and only one DefectArea stored in the foreign key column **DefectAreaId**; that is, this is a many-to-one case.

Property Name	Value
Association	DefectArea
Entity	DefectArea (tutorial-model)
Java Class	DefectArea
Paging Size	1

4. Right-click the newly added node **Child (Data Slice Type) DefectArea (tutorial-model)**, Type: many-to-one and choose **New Child** → **Field (Data Slice Field Type)**.
5. In the Property pane, set the **Name**, in a first instance to **Cold**.



Setting up Virtual Fields and Dynamic Lists Of Values (DLOVs)

Repeat step 4 to create a **Field (Data Slice Field Type)** node and set the **Name** as in step 5 to create a collection that includes the following standard system columns:

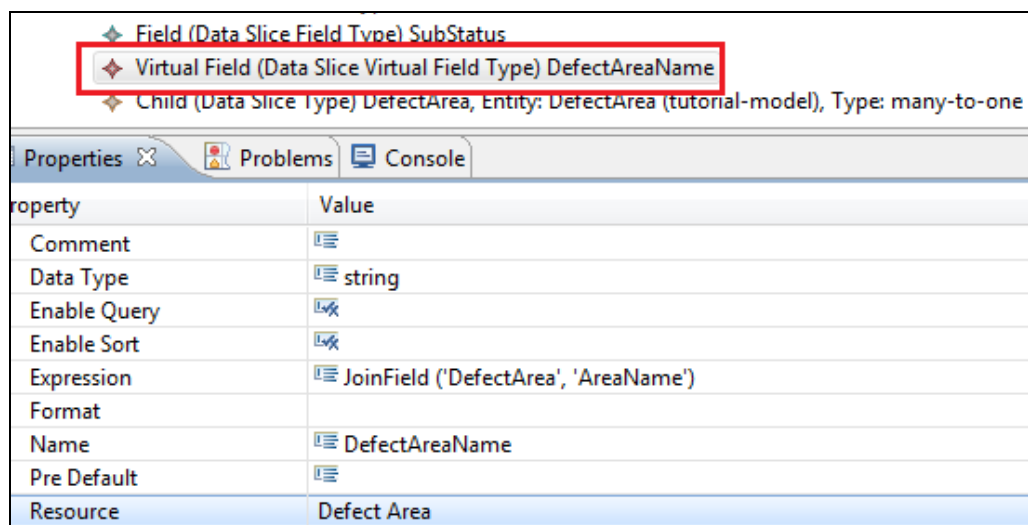
Cold
CoCreatedBy
CoCreatedDate
CoLastUpdBy
CoLastUpddate
CoRowVersionNum
AreaName

2. Add a Virtual Field to the Data Slice

Add the virtual field to the **AllDefect** slice.

1. Near the top of the tree, right-click the node **Root (Data Slice Type)** and choose **New Child → Virtual Field (Data Slice Virtual Field Type)**.
2. Set the following properties:

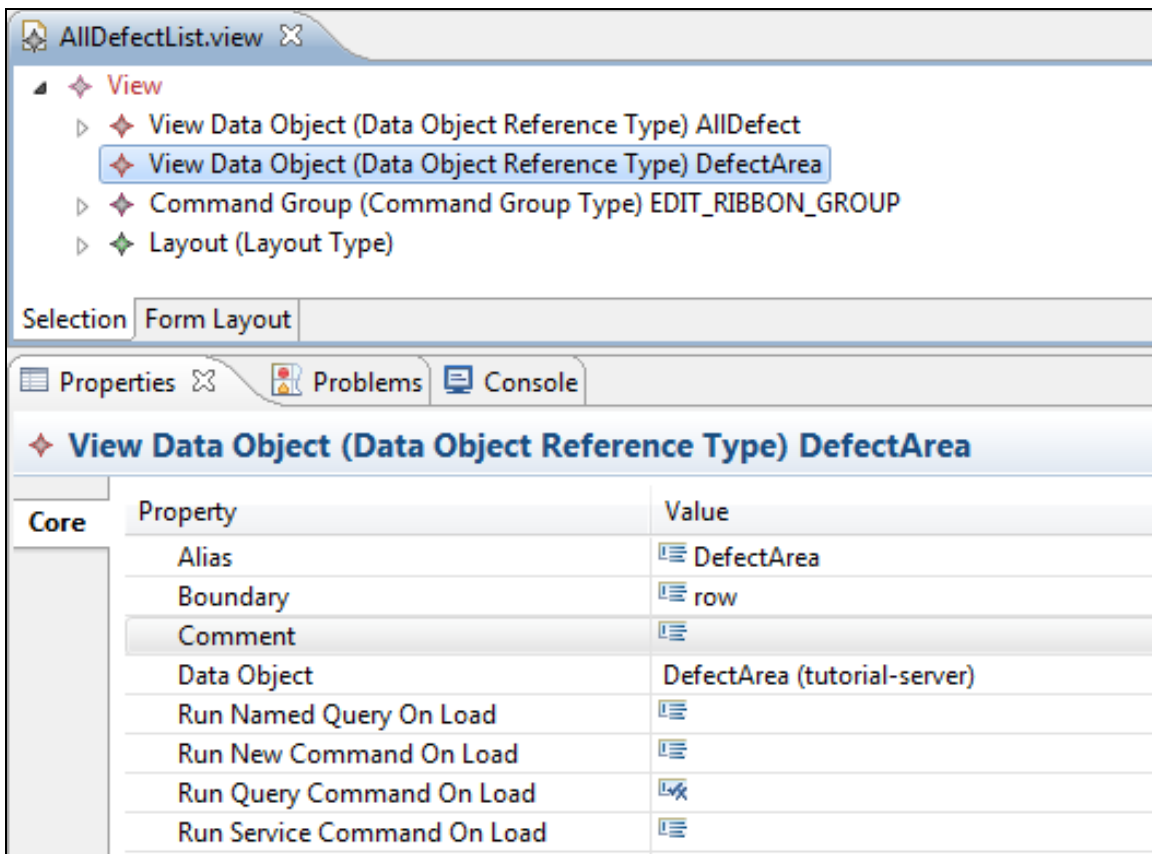
Property	Value	Notes
Data Type	string	
Expression	JoinField ('DefectArea', 'AreaName')	Press F2 when entering this field to display an intelligent text entry window in which you can press CTRL+ENTER to auto-complete your entry. The JoinField function inserts the AreaName of the DefectAreaID record from the DefectArea entity.
Name	DefectAreaName	This can be any name you like.
Resource	Defect Area	



3. Configure the Data Source for a Dynamic List of Values

Add a new data object to the view, to act as the data source for this dynamic list of values:

1. In the Metadata pane, under **tutorial-client** → **User Interface** → **View**, double-click **AllDefectList** to display **AllDefectList.view** in the Editor pane.
2. Right-click the top node **View** and choose **New Child** → **View Data Object (Data Object Reference Type)**.
3. In the Property pane, set **Data Object** to **Defect Area (tutorial-server)**, which is the data source for the dynamic list. Set **Alias** to **DefectArea**.

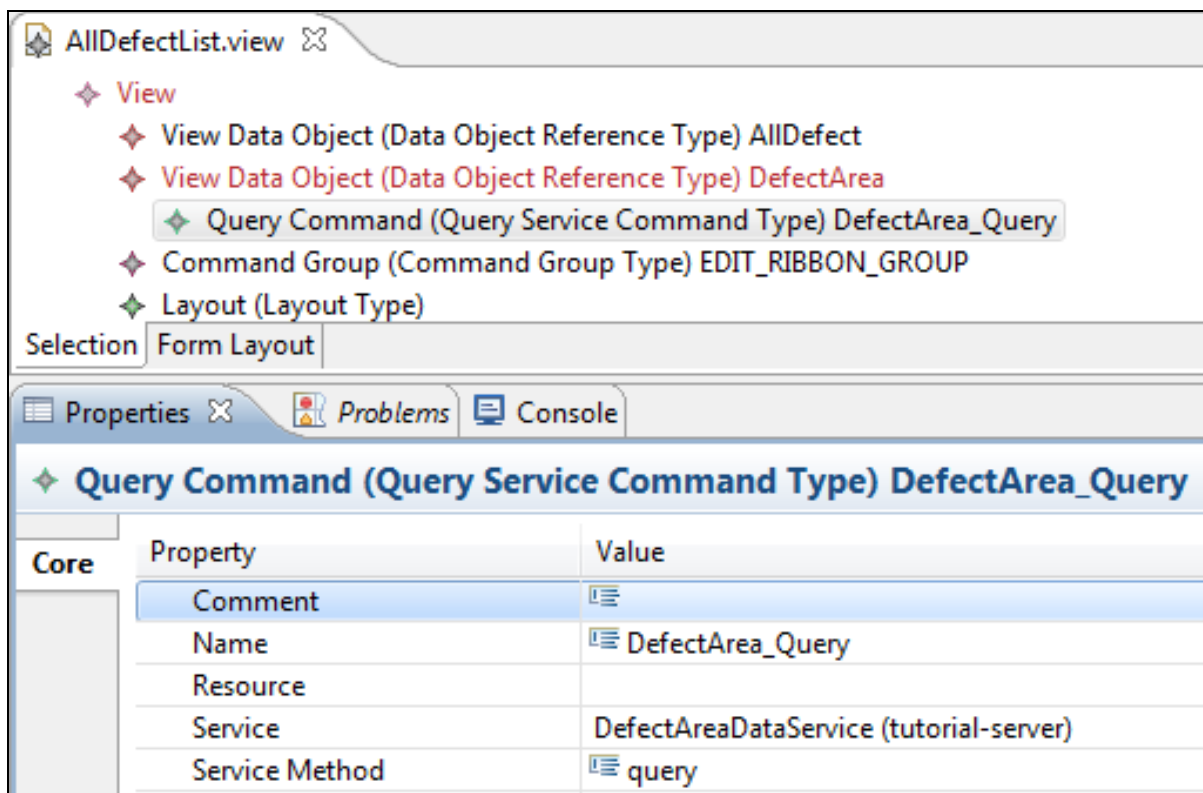


4. Right-click **View Data Object (Data Object Reference Type) DefectArea** and choose **New Child** → **Query Command (Query Service Command Type)**.

Setting up Virtual Fields and Dynamic Lists Of Values (DLOVs)

5. In the Property pane, set the following properties:

Property	Value	Notes
Name	DefectArea_Query	This can be any name you like
Service	DefectAreaDataService (tutorial-server)	This is to define which service and what method can query and give us the result of DefectArea
ServiceMethod	query	The method from the selected service you want to query for DefectArea



6. Right-click the new node **Query Command...** and choose **New Child → Input Binding (Input Binding Type)**.

Setting up Virtual Fields and Dynamic Lists Of Values (DLOVs)

7. Set the properties for **Input Binding...** to the following:

Property	Value
Data Object	DefectArea This defines the data object for the query; you will use the current query spec the user has entered in the query panel in the UI. Note: Because this view has no query panel displayed in the UI, this query spec will always be empty unless you have some custom coding that changes the query spec before the query happens.
Input Name	querySpec This is defined in the service generated by CodeObjects.

The screenshot shows the 'AllDefectList.view' editor. The 'View' tree on the left includes 'View Data Object (Data Object Reference Type) AllDefect', 'View Data Object (Data Object Reference Type) DefectArea', 'Query Command (Query Service Command Type) DefectArea_Query', 'Input Binding (Input Binding Type) querySpec' (highlighted), and 'Command Group (Command Group Type) EDIT_RIBBON_GROUP'. The 'Selection' tab is active, showing the 'Form Layout' section. Below this, the 'Properties' tab is active, displaying the 'Input Binding (Input Binding Type) querySpec' configuration table.

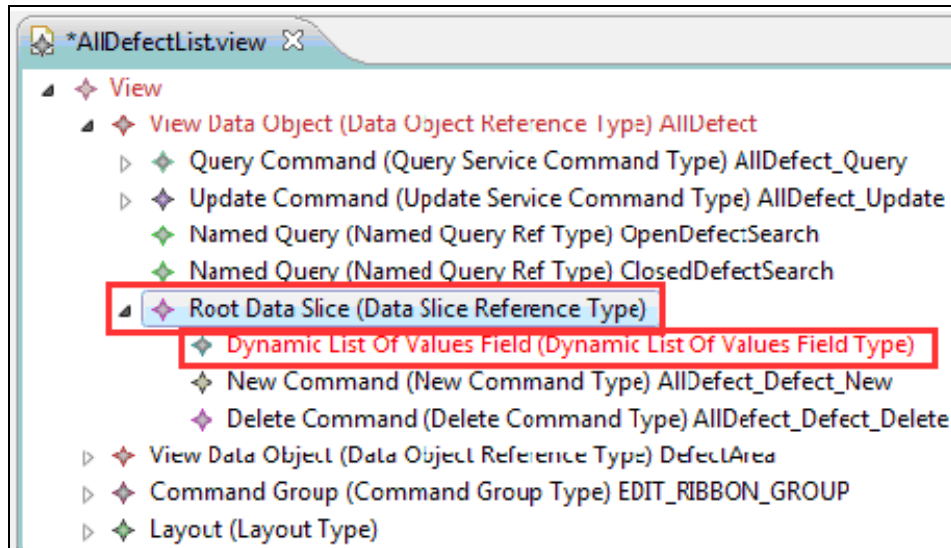
Core	Property	Value
	Comment	
	Data Object	DefectArea
	Input Name	querySpec
	Local Data	

Setting up Virtual Fields and Dynamic Lists Of Values (DLOVs)

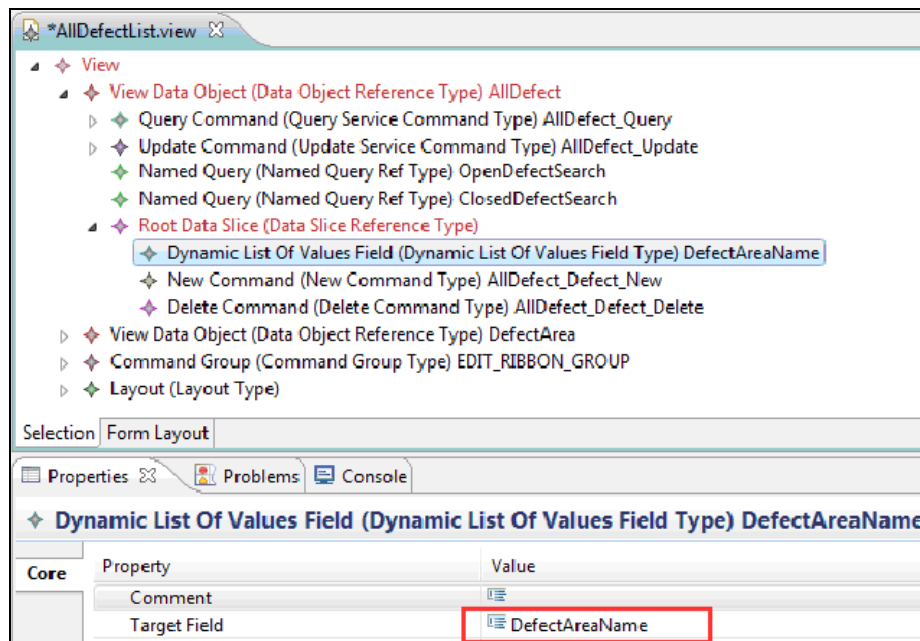
4. Create the Dynamic List Metadata

Create the dynamic list metadata.

1. Under **View Data Object (Data Object Reference Type) AllDefect**, right-click **Root Data Slice (Data Slice Reference Type)** and choose **New Child → Dynamic List Of Values Field (Dynamic List Of Values Field Type)**.



2. In the Property pane, set **Target Field** to **DefectAreaName**. This is where the dynamic list of values (DLOV) should appear. In this case, you are using the virtual field that you created earlier and will use this field to bind to this DLOV.



3. Right-click the node you just created and choose **New Child → Source Field (Field Binding Type)**. In the Property pane, set **Data Object** to **DefectArea** and **Field Name** to **AreaName**.

This describes the data source of DLOV. In this case, the DLOV data comes from the Data Object DefectArea's field AreaName.

The screenshot shows the 'AllDefectList.view' interface. The 'View' tree on the left includes a 'Dynamic List Of Values Field (Dynamic List Of Values Field Type) DefectAreaName' node, which has a child 'Source Field (Field Binding Type) AreaName' highlighted with a red box. The 'Properties' pane on the right shows the configuration for this field, with a table of properties and values highlighted by a red box.

Property	Value
Comment	
Data Object	DefectArea
Field Name	AreaName
Path	

4. Right-click the node **Dynamic List of Values Field...** and choose **New Child → Data Map (Data Object to Data Object Data Map Type)**. In the Property pane, set **Destination Name** to **AllDefect** and **Source Name** to **DefectArea**.

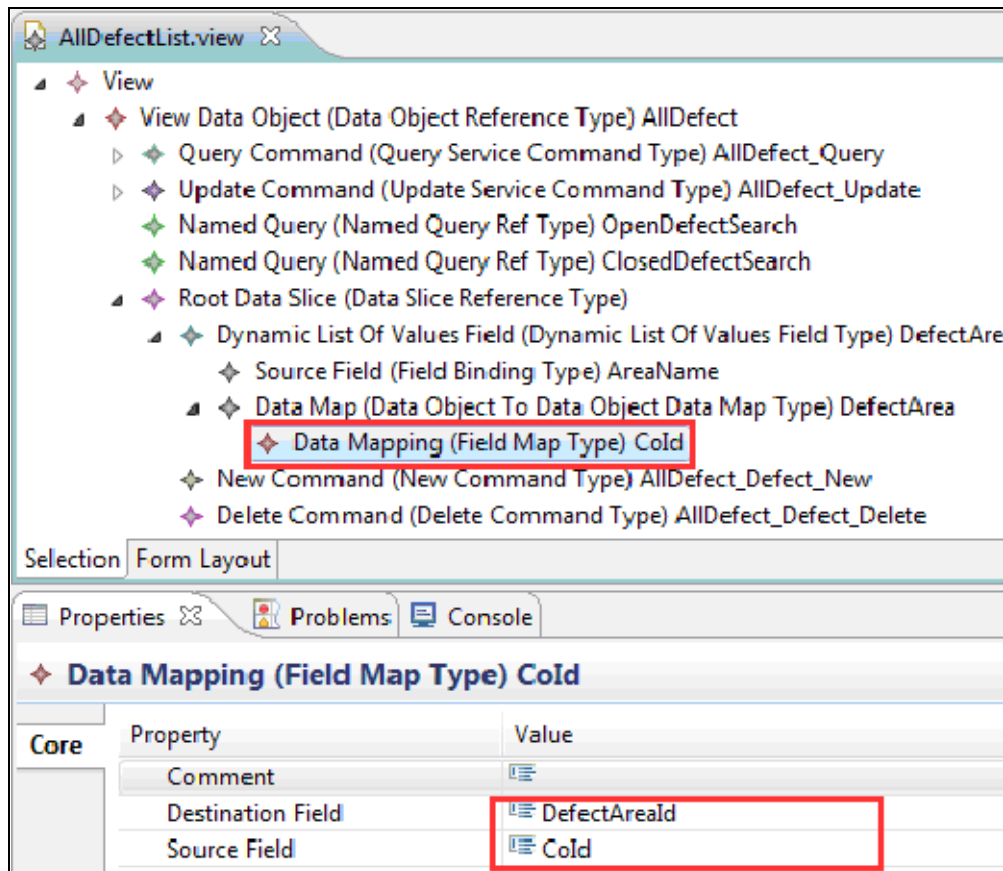
This Data Map specifies which data object, data slice, and field you should update for the user's selection from the DLOV. In this case, you update the data in Data Object AllDefect from the source DefectArea data object. Leave the **Path** empty because the field you set is at root level.

The screenshot shows the 'AllDefectList.view' interface. The 'View' pane on the left displays a tree structure of data objects and commands. The 'Data Map (Data Object To Data Object Data Map Type) DefectArea' node is highlighted with a red box. The 'Properties' pane on the right shows the configuration for this Data Map, with a red box around the 'Core' section.

Property	Value
Comment	
Destination Name	AllDefect
Destination Path	
Source Name	DefectArea
Source Path	

5. Right-click the new node **Data Map...** and choose **New Child → Data Mapping (Field Map Type)**. In the Property pane, set **Destination Field** to **DefectAreaId** and **Source Field** to **Cold**.

This is used together with the above Data Map. This indicates that the application update DefectAreaId in the AllDefect Data Object with the value from Cold in DefectArea.



5. Change the UI Control to Use the Dynamic List

Now you must change the UI to use the dynamic list control. You have created a virtual field in the **AllDefect** data object; that is **DefectAreaName**, which is also the target of your dynamic list. But there is an existing **DefectAreaId** field, which is a foreign key column that you should not expose to the end user.

1. In the Editor pane, expand the tree from **Layout (Layout Type)** through **Control (List Control Type)**. At the bottom of the tree (so far), right-click **Inline Form (Inline Form Type)** and choose **Open Layout Editor**.

Setting up Virtual Fields and Dynamic Lists Of Values (DLOVs)

2. Click the **(DefectAreaId)** label and press DELETE, and do the same to its control field to remove them.

The screenshot shows a form titled "AllDefectList.view". The form contains several fields arranged in a grid. The field "(DefectAreaId)" is highlighted with a red rectangle, indicating it is the target for deletion. Other fields include "Date Opened:", "Due Date:", "Closed Date:", "Assigned Date:", "Created By:", "Severity:", "Priority:", "Status:", "Sub Status:", "(Description)", "(ClientImpact)", and "(Resolution)".

3. In the Palette on the right, click **Label & Field (H)** to expand it. In the Palette, click **DefectAreaName**, and then click the empty spaces where you just deleted the (DefectAreaId) label and field.

The screenshot shows the same form as before, but with the "DefectAreaName" field added to the "Label & Field (H)" section of the palette on the right. A red arrow points from the "DefectAreaName" field in the palette to the empty space where the "(DefectAreaId)" field was previously located. The palette also shows other fields like "CoLastUpdate", "CoRowVersionNum", "DefectAreaId", "Description", "DueDate", "OpenDate", and "OwnedById".

Setting up Virtual Fields and Dynamic Lists Of Values (DLOVs)

4. Verify the result. Click the field **DefectAreaName** and notice that the Property pane indicates a DLOV Control Type, which is what you should expect.

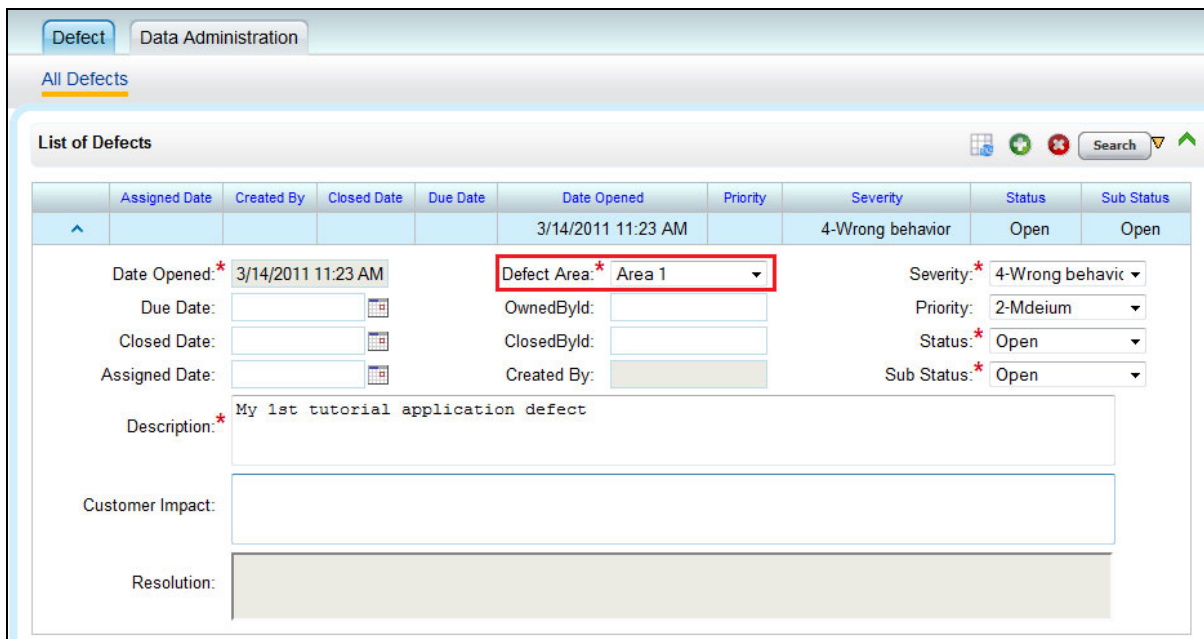
The screenshot displays a software interface with a form layout and a properties pane. The form layout includes fields for Date Opened, Due Date, Closed Date, Assigned Date, Defect Area, Owned By Id, Closed By Id, Created By, Description, Customer Impact, and Resolution. The Defect Area field is highlighted with a red box. A red arrow points from the Defect Area field to the Properties pane. The Properties pane shows the DLOV Control Type for DefectAreaName.

Core	Property	Value
	Command	
	Comment	
	Data Slice Path	
	Field Name	DefectAreaName
	Horizontal Alignment	Left
	Read Only Override	
	Required Override	
	Vertical Alignment	Bottom

Setting up Virtual Fields and Dynamic Lists Of Values (DLOVs)

Test the Result

1. Ensure that all changes are saved. Click **File** → **Save All**.
2. Archive, deploy, and run the client.
3. Click the **New**  button and notice the changes, especially the new virtual field **Defect Area** and its dynamic pull-down list.



Defect Data Administration

All Defects

List of Defects

Assigned Date	Created By	Closed Date	Due Date	Date Opened	Priority	Severity	Status	Sub Status
				3/14/2011 11:23 AM		4-Wrong behavior	Open	Open

Date Opened: * 3/14/2011 11:23 AM Defect Area: * Area 1 Severity: * 4-Wrong behavior

Due Date: OwnedByld: Priority: 2-Medium

Closed Date: ClosedByld: Status: * Open

Assigned Date: Created By: Sub Status: * Open

Description: * My 1st tutorial application defect

Customer Impact:

Resolution:

Review

In this tutorial, you:

- Created the DefectArea data table.
 - Created an entity for the DefectArea data table.
 - Created a required field in the DefectArea entity.
- Defined relationships between the entities.
 - Defined the Link Key to the DefectArea in the Defect entity.
 - Defined the many-to-one association between the Defect Entity and the DefectArea Entity.
 - Define the one-to-many Association from the DefectArea Entity to the Defect Entity.
- Created the view for Defect areas and Employee data.
- Created a new widget and added it to the client.
- Created a virtual field with a dynamic list.
 - Added a data slice to the data object and added a virtual field to the data slice.
 - Configured the data source for a DLOV and created the dynamic list metadata.
 - Changed the UI control to use the dynamic list.