Using OptimalJ: Tutorials

OptimalJ 3.1

OptimalJ 3.1
Using OptimalJ: Tutorials

Trademarks

OptimalJ 24-hour online information and support

For using the OptimalJ suite of products, two Web sites are available. JavaCentral, at *http://javacentral.compuware.com*, lets you communicate with other Java developers through forums. You will also find FAQ's, technical tips, news and other information to help you to build your applications. Support for the JavaCentral forums is available to anyone. Register online at *http://javacentral.compuware.com*.

Customers who have purchased OptimalJ with a support contract can use FrontLine at *http://frontline.compuware.com*. This provides access to online support information including product documentation and tutorials, up-to-date platform availability, incident submission and a list of known software problems.

For the latest version of the documentation always check the FrontLine site.

Your suggestions and comments about OptimalJ documentation are highly valued. Please send your reactions to:

Compuware Europe B.V. P. O. Box 12933 1100 AX Amsterdam The Netherlands

e-mail address: admin@optimalj.com

Contents

1 Tutorials

Chapter 1 Tutorials

The tutorials provide step-by-step instructions describing the various aspects of building an application with OptimalJ.

General tutorials

Domain level tutorials

Application level tutorials

Integration level tutorials

Code level tutorials

Meta model level tutorials

1.1 Your first OptimalJ application

In this tutorial, you will create a simple application with two classes: *Salesorder* and *Orderline*.

Creating a application involves:

- Creating a domain package that contains domain classes with domain attributes
- Creating a domain association between the two domain classes

- Generating all the models and code and compiling the application
- Setting up the database to test the application

The generated application allows you to create, retrieve, update, and delete Salesorders and Orderlines.

Prerequisites

None

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial, you learn how to:

- Create a model package, domain class, and domain class association
- Create application models from the domain model
- Generate and compile code
- Create database tables
- Test the application

Step 1 ? Prepare the filesystem

In your file system, create the following directories:

1. `\OptimalJ\myApplication\myAppModel`?this directory holds your model definitions and the code generated for creating the DBMS.
2. `\OptimalJ\myApplication\myAppEjbCode`?this directory contains the EJB application code.
3. `\OptimalJ\myApplication\myAppWebCode`?this directory contains the Web application code.

Step 2 ? Create a new project

To create a new project:

1. From the menu, select **Project>New OptimalJ Project**.
2. Set the project name to `MyFirstProject` and click **Next**.
3. Select the type of project. Select `New Model` and browse to `\OptimalJ\myApplication\myAppModel`. Click **Open** and then **Next**.

4. Create a package structure. Enter `ordersample` as the fully qualified package name.

   Set the initial package structure to `Three Tier Application Structure` and click **Next**.

Note: By selecting a three-tier initial package structure, you create a structure containing a domain model with a class model, and an application model with a database, an EJB model and a Web model. These models contain the appropriate Technology Pattern to generate elements for a three-tier application (Web components, EJB components and a database schema).

5. Configure mount point settings. The EJB and Web code is generated in separate directories (or mount points). Keep `Mount each filesystem yourself` selected, click **Next**, and then **Finish**.

Step 3 ? Inspect the created model packages

1. In the Explorer [Domain Model], select `ordersample` to see the nodes.

Figure 1-1 Domain model packages

2. In the Explorer [Application Model], select `ordersample` to see the nodes.

Figure 1-2 Application model packages

3. In the Explorer [Code Model], select the elements created in the code model. As you can see, two type of objects are created:

- Folders to hold the files for the models.
- Files with the extension `.xcm` to hold the details of the elements of the models.

Figure 1-3 Code model view

Step 4 ? Create the Salesorder domain class

The domain models in OptimalJ hide implementation details and are used to generate other models or code. The domain model contains two models: *class* and *service*. OptimalJ generates application model elements from these domain models.

To create a `Salesorder` class in the domain class package:

1. In the Explorer [Domain Model], right-click the `domain.class` package and select **New Child>DomainClass** to start the Create DomainClass wizard.
2. In the **Name** field, enter `Salesorder` and click **Next**.

Figure 1-4 Create Domain Class wizard: Enter Domain Class Name

3. Add attributes to `Salesorder` as shown in the illustration below. To add an attribute, click the **Add** button and enter or select the appropriate values for the new attribute. Click in the attribute's **Type** field to display a menu of possible type values.

Figure 1-5 Create Domain Class wizard: Enter Attributes

4. Click **Finish**, ignoring the other wizard pages. This closes the wizard and creates the class `Salesorder` in the `domain.class` model.

   The result in the Explorer [Domain Model] looks like this:

Figure 1-6 Domain model with Salesorder

Step 5 ? Create the Orderline domain class

To create the `Orderline` class in the domain class package:

1. In the Explorer [Domain Model], right-click the `domain.class` package and select **New Child>DomainClass** in the pop-up menu to display the Create DomainClass wizard.
2. In the **Name** field, enter `Orderline`, and click **Next**.

Figure 1-7 Create Domain Class wizard - Enter Name

3. Add attributes to `Orderline` as shown in the illustration below. To add an attribute, click the **Add** button and enter or select the appropriate values for the new attribute. Click in the attribute's **Type** field to display a menu of possible type values.

Figure 1-8 Create Domain Class wizard: Enter Attributes

4. Click **Finish**, ignoring the other wizard pages. This closes the wizard and creates the class `Orderline` in the `domain.class` model.

   The result in the Explorer [Domain Model] looks like this:

Figure 1-9 Domain model with Salesorder and Orderline

Step 6 ? View the Domain Class Diagram

To view the UML domain class diagram for your domain model:

1. In the Explorer [Domain Model], double-click the `domain.class` model package.

   This opens the domain class diagram in the **class** tab of the Source Editor.

Figure 1-10 Domain class diagram

Note: Use the toolbar at the top of Source Editor [class] to zoom in or out of the diagram. A miniature version of the domain class diagram can be viewed in the Explorer [Diagram Thumbnail]. (If this is not visible, choose ***View>Diagram Thumbnail****.) The thumbnail view always shows your entire diagram, and shows the rectangle you are currently viewing in the diagram editor if you are viewing only part of the diagram. You can drag this rectangle to view another part of the diagram.*

Step 7 ? Create an association between Salesorder and Orderline

`Salesorder` must have a composition association with `Orderline` where `Salesorder` is the composite class *composed of* the part class `Orderline`. Because Salesorder is composed of Orderline, an Orderline can only be associated with one Salesorder and cannot exist without an association to a Salesorder. Domain associations can be created in the Explorer [Domain Model] or in the domain class diagram. The last method is used in this tutorial.

To create the association between `Salesorder` and `Orderline` class, perform the following steps in the domain class diagram:

1. In the toolbar at the side of Source Editor [class], click the icon to create an *association*.

   With the tool selected, click inside the class Salesorder, drag the mouse to the class Orderline, and release the mouse.

   The Create Domain Association wizard now appears.
2. In the pane **Define Association End 1**, keep **Role Name** as `Salesorder`.

   Set **Aggregation** to **Composite**. OptimalJ sets **Multiplicity** to **Exactly one**.

   Click **Next**.
3. In the pane **Define Association End 2**, keep **Role Name** as `Orderline`.

   Keep **Aggregation** as none.

   Set **Multiplicity** to zero or more. This means that a Salesorder can have no or more then one Orderline.

   Click **Next**.
4. In the pane **Enter Name**, keep **Name** as `Salesorder_Orderline`, and click **Finish**.
5. The diagram now contains the association.

Figure 1-11 Domain class diagram: Salesorder is the composite class associated with the part class Orderlines

Step 8 ? Create the DBMS model from the domain model

The DBMS model contains the definitions for relational database tables used by the application. SQL scripts for creating the tables in the database are generated from this model.

Note: In this tutorial, you generate the application models one by one. However you can generate all models in one action by choosing ***Model>Update all models****.*

To create the DBMS model:

1. From the menu, select **Model>Generate Model>Generate Application Models>Generate DBMS from Domain** to display the Generate DBMS Model from Domain Model wizard.

Figure 1-12 Generate DBMS Model from Domain Model

2. Select `ordersample.domain` and click **Next**.
3. The wizard displays all packages within the application model for the current project.

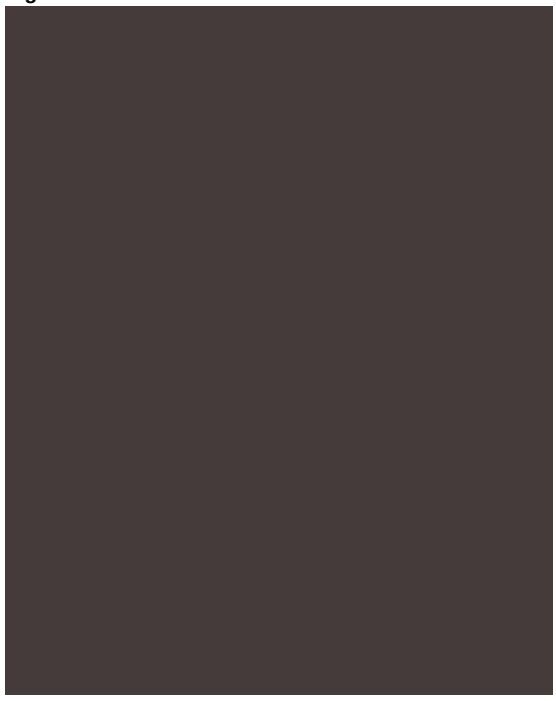
Figure 1-13 Generate DBMS Model from Domain Model

Select `ordersample.application.dbms` and click **Finish**. This generates your database model from your domain model, creating relational data schema and table definitions.

Note: When you execute this step, OptimalJ checks the integrity of your domain model. OptimalJ always checks model integrity before it generates models or code.

4. In the Explorer [Application Model], view the generated DBMS model:

Figure 1-14 Database model



Step 9 ? Create an EJB model from the domain model

The EJB model contains the model information for entity and session components that will be used to generate entity and session beans. In this step, you create entity components that will be used to generate entity beans to access data in the database tables.

To create the EJB model:

1. From the menu, choose **Model>Generate Model>Generate Application Models>Generate EJB from Domain** to display the Generate EJB Model from Domain Model wizard.

Figure 1-15 Generate EJB Model from Domain Model

2. Select `ordersample.domain` and click **Next**.
3. The wizard displays all packages within the application model for the current project.

Figure 1-16 Generate EJB Model from Domain Model

Select `ordersample.application.ejb` and click **Finish**. This generates your EJB model from your domain model, creating component and data schema definitions.

4. In the Explorer [Application Model] view the generated model:

Figure 1-17 EJB model

Note: OptimalJ does not create an entity component or a data schema for Orderline because the association between Orderline and Salesorder is defined as composite. The EJB entity component and EJB data schema for Salesorder contain elements for Orderline.

Step 10 ? Create a Web model from the domain model

The Web model contains the definitions for the Web components. The Web model is used to generate JSP, HTML, and Servlet code.

To create the Web model:

1. From the menu, choose **Model>Generate Model>Generate Application Models>Generate WEB (EJB based) from Domain** to display the Generate Web Model from Domain Model wizard.

   The wizard displays all packages within the current project.

Figure 1-18 Generate Web Model from Domain Model

2. Select `ordersample.domain` and click **Next**.
3. The wizard displays all packages within application model for the current project.

Figure 1-19 Generate Web Model from Domain Model

Select `ordersample.application.web` and click **Finish**. This generates your Web model from your domain model, creating component and data schema definitions.

4. In the Explorer [Application Model] view the generated model:

Figure 1-20 Web model

Note: Because the association between Orderline and Salesorder is defined as composite, the WEB component and WEB data schema for Salesorder contain the elements for Orderline.

Step 11 ? Generate the code

After completing all modeling activities, you can generate the code for the application components.

To generate the code:

1. From the menu, choose **Model>Generate All Code**.

Note: OptimalJ checks the integrity of your models before generating code.

2. You are prompted to mount a directory where the code for the EJB module can be generated. The EJB module contains the code that is executed by the application server.

   In the wizard Select Filesystem to generate Code for Module:ejb, click **Mount New Filesystem...**.

3. In the wizard New Wizard, select `Local Directory` and click **Next**.
4. Browse to `\OptimalJ\myApplication`, select `myAppEjbCode`, and click **Finish**.
5. In the wizard Select Filesystem to generate Code for Module:ejb, select the directory `\OptimalJ\myApplication\myAppEjbCode` and click **OK**.
6. You are then prompted to mount a directory where the code for the Web module can be generated. The Web module contains code that is executed by the Web server.

   In the wizard Select Filesystem to generate Code for Module:web, click **Mount New Filesystem...**, to mount the directory `\OptimalJ\myApplication\myAppWebCode`.
7. In the wizard New Wizard, select `Local Directory` and click **Next**.
8. Browse to `\OptimalJ\myApplication`, select `myAppWebCode` and click **Finish**.
9. In the wizard Select Filesystem to generate Code for Module:web, select the directory `\OptimalJ\myApplication\myAppWebCode` and click **OK**.
10. Wait for code generation to complete. The message line in the main window reports how the generation is progressing. The Generator tab in the Output Window displays `Finished generating all code` when generation is complete. A database for code completion is also created. This database is used to auto-complete the methods you edit, while modifying the generated code.
11. Examine the results by opening the folders and files in the Explorer [Code Model] window.

Figure 1-21 Examine the results

Step 12 ? Compile the generated code

After generating code for the application, the code must be compiled.

To compile the code:

1. Choose **Project>Compile Project**.
2. Wait for the message Finished Project MyFirstProject in Output Window [Compiler]

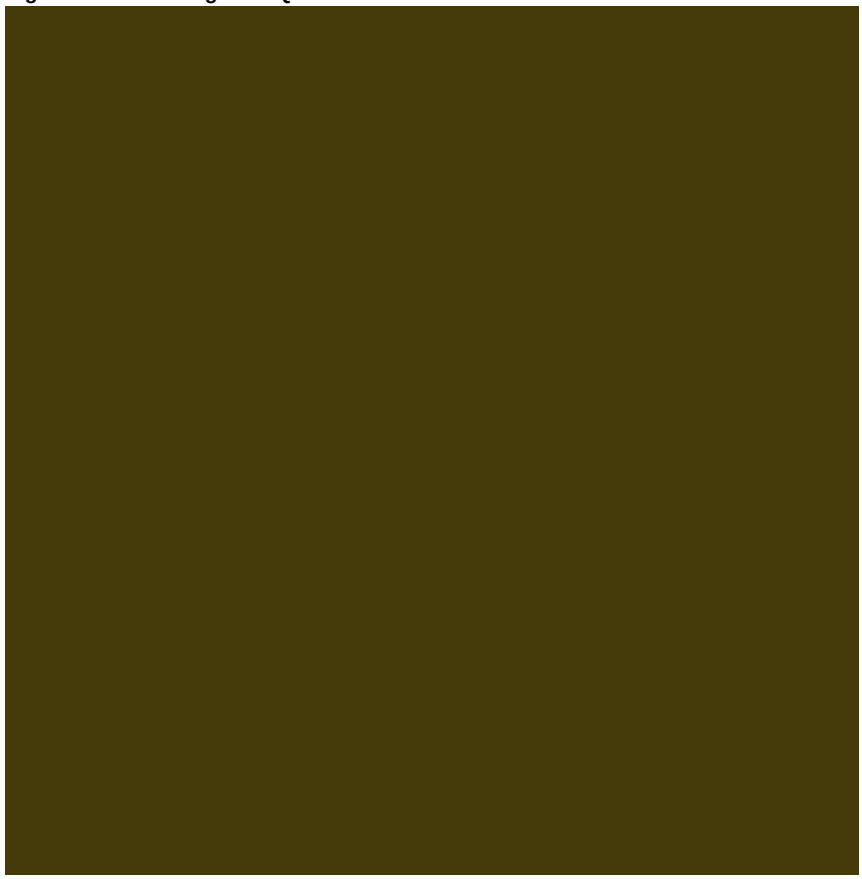
Step 13 ? Create tables in the database

Before you can run the application, you must create the database tables. The DBMS package contains a DBMS *metafile* and all SQL scripts needed to drop, create, and initialize tables. In this example, you use a SOLID database, and the database must be empty.

To create the tables:

1. Start the default SOLID database server delivered with OptimalJ, by executing the SOLID shortcut installed on your desktop. The database contains sample data for the CRM application. Because you are using other tables than used by the CRM example, using this database should give no problem.
2. In the Explorer [Code Model] open `ordersample\application\dbms`.
3. Right-click `Solid_MetaOrdersample.sqm`. This file contains connection data allowing OptimalJ to connect to your database.
4. Choose **SQL Workbench in context**.

Figure 1-22 Starting the SQL Workbench



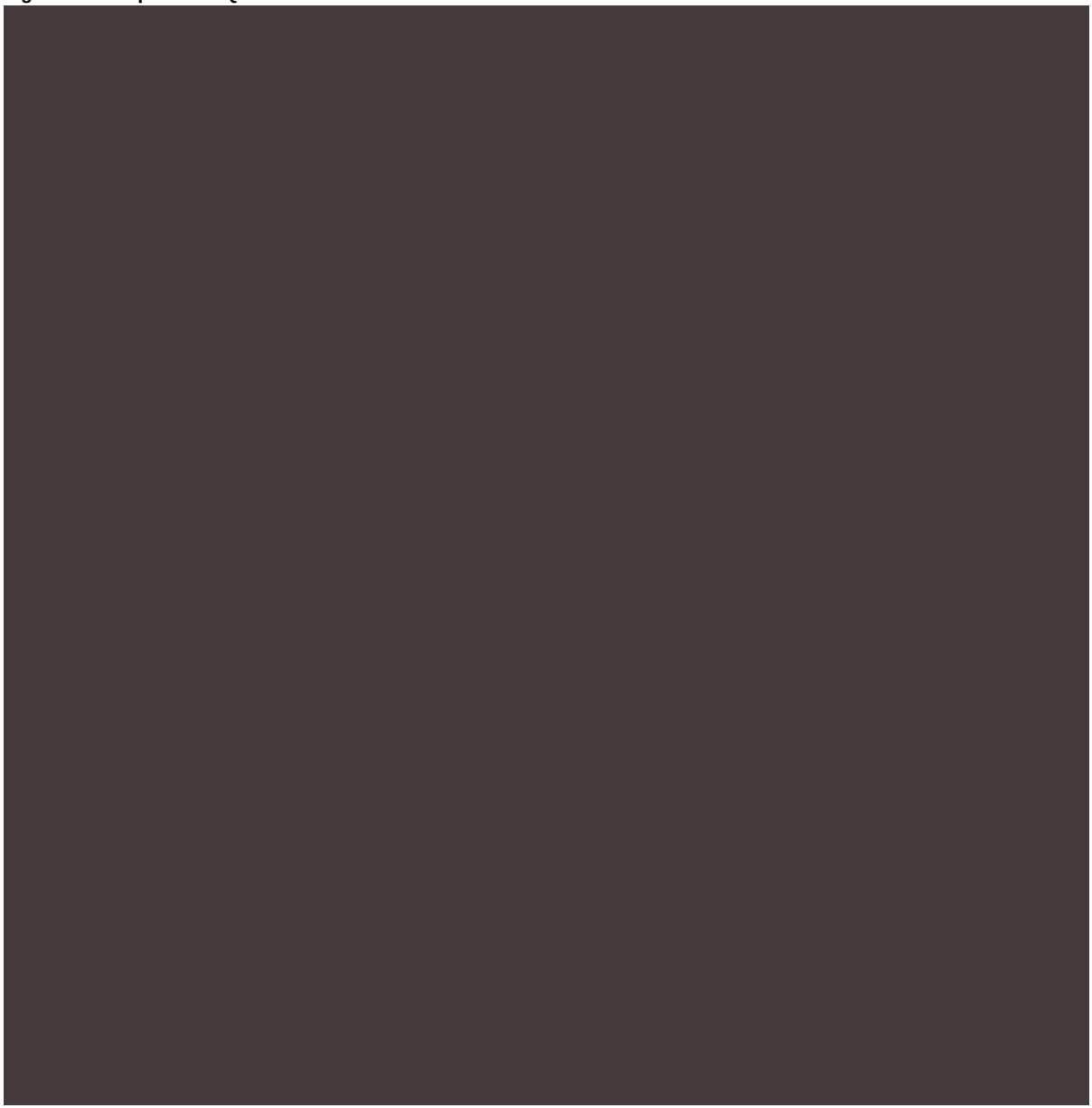
The OptimalJ SQL Workbench starts and shows the Connect window. If you have properly configured your SOLID database

settings, you can accept the default field values and click **OK**. For more information, see *Setting up a SOLID database*.

Figure 1-23 Connect window

5. Click **Create** in the OptimalJ SQL Workbench window. This loads the table generation SQL script into the command window. If the tables already exist, you can load the SQL script to drop the tables by clicking **Drop**.
6. Click **Exec Batch** in the OptimalJ SQL Workbench window. The **<workspace>** pane shows the result of the execution of the SQL commands.

Figure 1-24 OptimalJ SQL Workbench



7. Close the SQL Workbench (**File>Close**).

Step 14 ? Start the Application Server

To run the application, you need to start the application server integrated with OptimalJ.

To start the Application Server:

1. From the menu, select **Test>Start Application Server**.
2. OptimalJ starts JBoss, which is the default EJB Server for the integrated testing environment.
3. After JBoss has been started, Tomcat is started. TomCat is the default Web server for the integrated testing environment.
4. After Tomcat has been started, OptimalJ starts the default Web browser, displaying the MainMenu page.

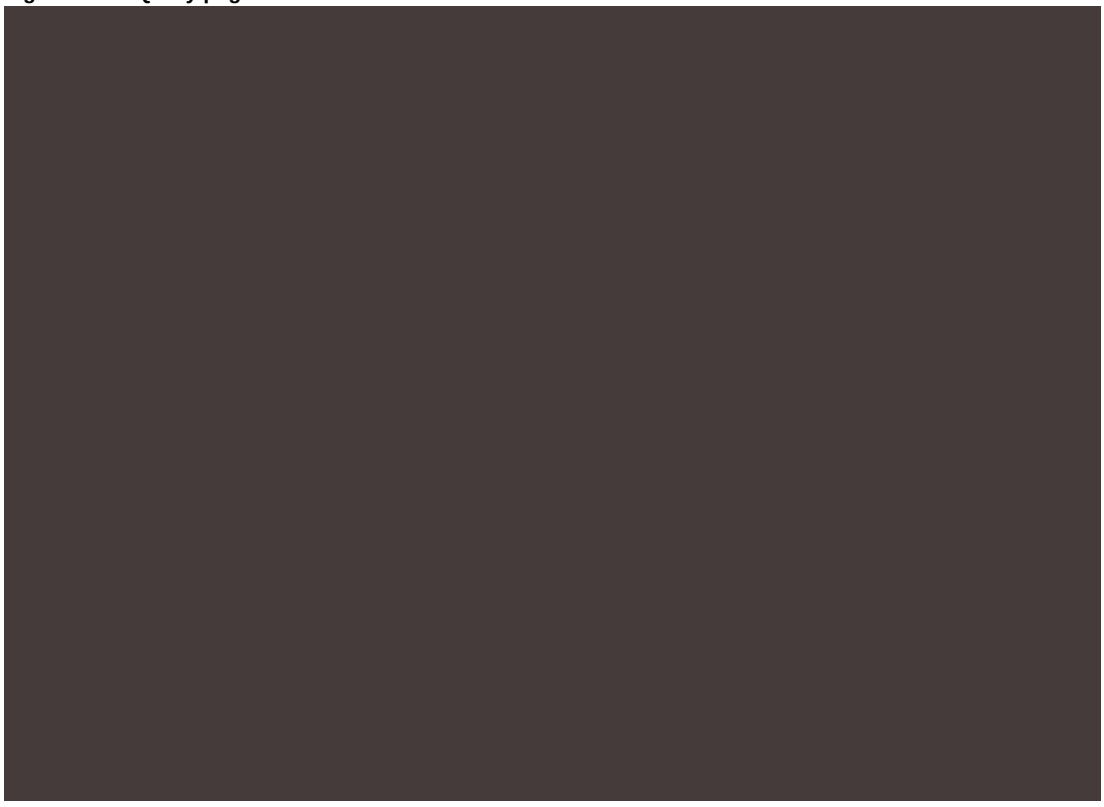
Step 15 ? Test the application

The MainMenu page is the home page of an OptimalJ Web application.

To test the application:

1. In the MainMenu, click **Maintenance Salesorder** to display the query page.

Figure 1-25 Query page



2. In the menu bar, click **New** to create a new record. The create Salesorder page appears.
3. Enter Salesorder data in the fields as shown in the following illustration, but do not click **OK** yet.

Figure 1-26 Create new Salesorder

Note: The Delete button is disabled, because a new Salesorder is being created.

4. Click **Create** to open a new page to create a new Orderline.
5. Enter Orderline data as shown in the following illustration and click **OK**.

Figure 1-27 Create new Orderline

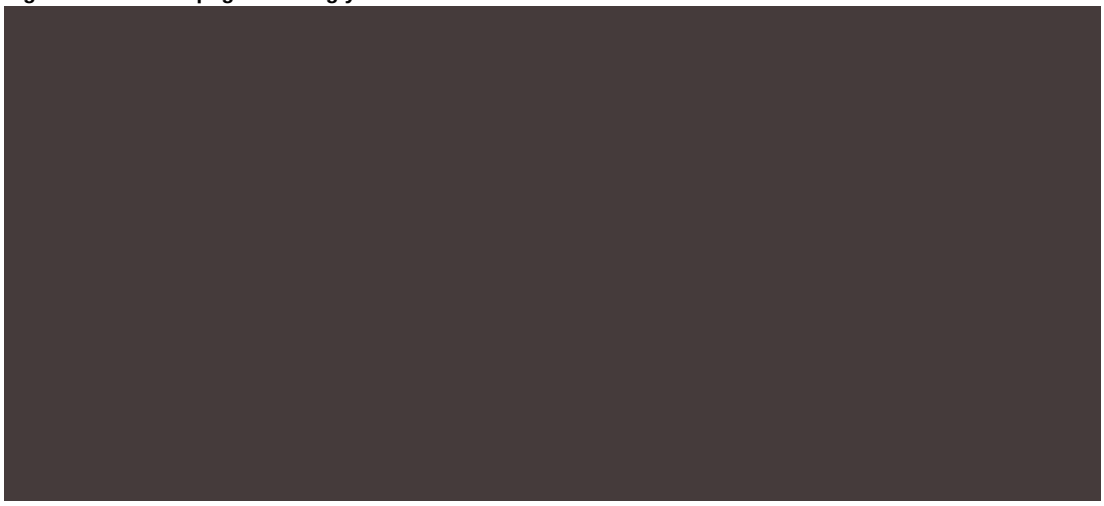
6. In the create Salesorder page, the Orderline has been added.

Figure 1-28 Salesorder with Orderline

Click **OK** to display the browse page.

7. Click **Submit** to finalize the transaction with the database. The query page is displayed.

Figure 1-29 Store page showing your records



8. In the query page, click **Browse** (with the default % in the **uniqueid** field) to display the Salesorder information in the database.

Figure 1-30 Browse result

9. After you have completed testing, choose **Test>Stop Application Server** to stop the application.
10. Stop the Solid database server, which is the DBMS server used in this tutorial. If you are using your own database, stop the DBMS

server as described in the documentation for your DBMS. (Stop the Application server first, as it may attempt to access the DBMS as it shuts down.)

In this tutorial, you performed all the steps needed to create a simple application. This application allows you to retrieve, create, update, and delete `Salesorders` and `Orderlines` and is fully compliant with the J2EE architecture. To support the development process, OptimalJ generated all the code needed for your application components and provided you with the appropriate tools to deploy and test your application.

1.2 Creating a new project

In this tutorial, you create a new OptimalJ project to build an application. An OptimalJ project allows you to distribute all the files of your project over a file structure that reflects the architecture you want to apply, for example, a two-tier, or a three-tier architecture. You can add model packages to refine the structure of the application. A project can contain several applications. If you want to add an application to your existing project, you have to mount a file system for the application manually. You create new projects using the New OptimalJ Project wizard.

Prerequisites

Nothing required.

Duration

This tutorial takes approximately 15 minutes to complete.

Objectives

In this tutorial, you learn how to create a new project.

Step 1- Prepare the filesystem

Create an empty directory `\OptimalJ\MySampleProject\MyModel`.

Step 2 - Set the project name

From the menu bar, select **Project>New OptimalJ Project** to display the New OptimalJ Project wizard. The wizard prompts you to enter a project name. Set the project name to `MySampleProject` and click **Next**.

Step 3 - Select the project name

Each project must have a directory available to hold the project models. If the directory is not available, you can either create the directory outside OptimalJ or create it during the directory mounting process. This tutorial demonstrates the use of directories created outside OptimalJ. Select `New Model`, browse to the directory `\OptimalJ\MySampleProject\MyModel` using the browse button, and click **Next**.

The Select the Type of Project pane shows three options for the project type:

- New model?you create a new model for your application.
- Mount existing model?you want to base your application on an existing model.
- Experiment with one or more example models?choose this option if you want to experiment with a *CRM Example Application* that OptimalJ provides.

Step 4 - Create a file structure

The design of your application requires an initial structure, for example a two-tier, or a three-tier architecture. In this tutorial, accept the default architecture, which is a three-tier structure with integration. You can provide a qualified package name to the initial structure. Accept the default `com.compuware.mysampleproject` in the Enter Fully Qualified Package Name and Properties pane, and click **Next**.

Note: The wizard pane also shows a field `Sub Model Name`. If you use model packages with equal top model package names, you need to qualify the model packages by a unique sub model name. See Model package structure for the details.

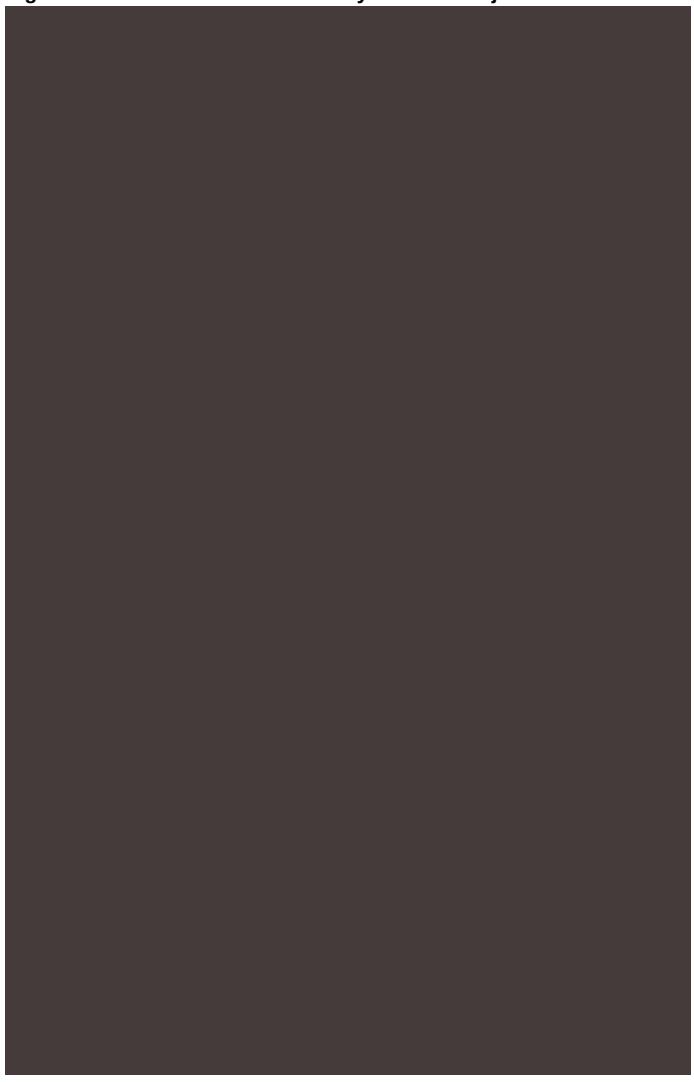
Step 5 - Set the automount settings

When you generate code from your models, OptimalJ needs mounted file systems to store the code. OptimalJ uses *Mount points* to address these systems. For a two-tier architecture, OptimalJ needs a single mount point to store the Web code. A three-tier model requires a mount point for the code generated from the Web model and a mount point for the code generated from the EJB model. In the Automount Settings pane you can choose to define these mount points yourself (default), or choose a base directory under which Optimal generates the mount points automatically. Accept the default `Mount each filesystem yourself` and click **Next**.

Step 6 - Select framework sources

The Include Source Code and Archives pane asks you to mount the archive `alturalib-src.zip`. The archive contains the Java source files of the OptimalJ framework. The archive is provided for instructional reasons. Accept the default option and click **Finish**. In the Explorer[Code Model] you can view the code model.

Figure 1-31 Code model of a newly created Project



In this tutorial you created a new project for a new application. You mounted a local directory to hold the project models. You defined the mount points for the generated code. You can now start to build your application in the domain model, update your models and generate the code. The tutorial *Developing your first OptimalJ application* demonstrates how you can take these steps.

Further reading

For more information, see also *OptimalJ Project* and other documentation topics on *OptimalJ environment and directories*.

1.3 Setting up a SOLID database

This tutorial guides you through the process of setting up a new SOLID database. Some of the tutorials included in the OptimalJ online help require you to create a new empty SOLID database.

A SOLID database server serves only one database at a time. The database being served is defined by a *folder* which is specified in the SOLID shortcut property called **Start in:** (Microsoft Windows). When the SOLID server is started, it uses the database located in that folder. By default, the database is in a file called `solid.db`. If, when the SOLID server is started, there is no database in the folder the server creates a new `solid.db` database.

An OptimalJ installation includes a pre-configured SOLID 4.0 server and a default database. The database contains the CRM database schema initialized with data. The CRM database is located in the folder `OptimalJInstallation\Solid_db`.

Prerequisites

- Familiarity with databases.

Duration

This tutorial takes approximately 30 minutes to complete.

Objectives

This tutorial describes how to:

- Create a new SOLID database for Windows (step 1) and Linux (step 2).
- Confirm OptimalJ's settings for SOLID (step 3).

Note: On Windows operating systems, follow the steps 1 and 3. On Linux operating systems, follow the steps 2 and 3.

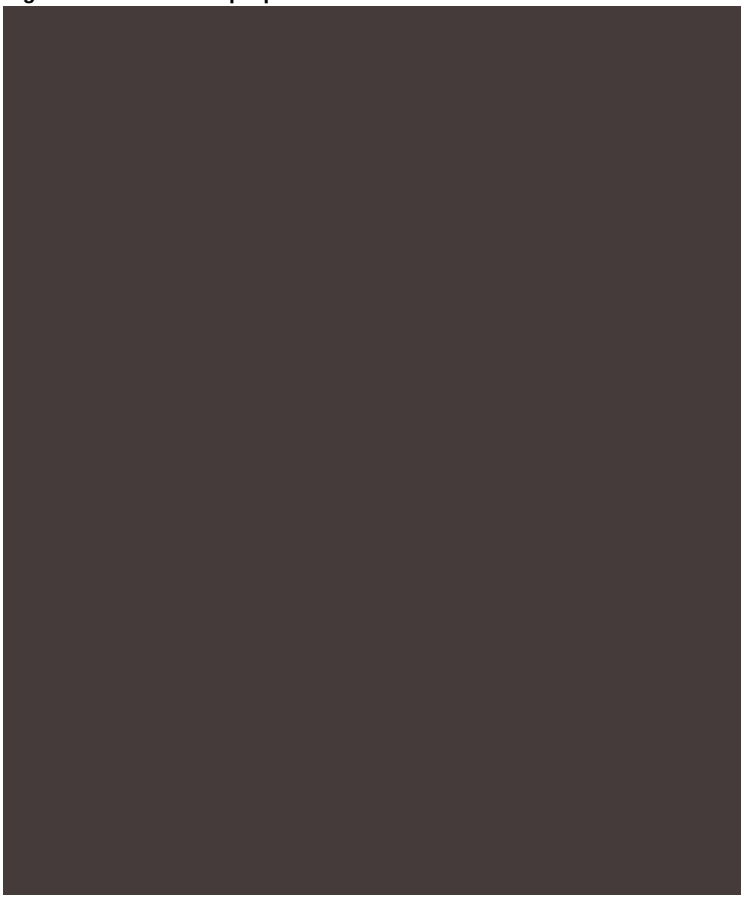
Step 1 - Create a new SOLID database (Windows)

1. In your file system, create a new folder for your SOLID database, for example,

   `OptimalJInstallation\SolidFolder`.

   Do *not* use the folder `OptimalJInstallation\Solid_db` because it contains the default CRM database.
2. Copy the SOLID license file `solid.lic` from `OptimalJInstallation\Solid_db` to the newly created `OptimalJInstallation\SolidFolder`.
3. Right-click the *Solid for OptimalJ* shortcut on the desktop and select **Properties** from the pop-up menu to display the properties dialog. Then select the *Shortcut* tab.

Figure 1-32 Shortcut properties



4. In the **Start in:** field, enter the path to the folder you created in Step 1, for example, `OptimalJInstallation\SolidFolder`. This is the location where the database is created.
5. Click **OK**, then close the dialog.
6. As the folder `SolidFolder` is newly created and does not contain a `solid.db` file, the SOLID server creates a new database when it starts. Double-click the *Solid for OptimalJ* shortcut to start the SOLID server.
7. In the Creating a new database dialog, create a new database using the listed properties, then click **OK**.

Table 1-1 SOLID database properties

Property	Value
System catalog	`optimalj`
Username	`optimalj`
Password	`optimalj`

Figure 1-33 Creating a new database

Note: The string `optimalj` *is a default value which reflects the default settings within OptimalJ. If you wish to use different values for the above properties you would have to change the same properties within OptimalJ.*

Note: The database created uses the default name solid and the default port 1313. If you are running another database concurrently using the same name on the same port, or if the port 1313 is already in use, you need to modify the `solid.ini` *file to change the name and the port settings. A default* `solid.ini` *file is located in the* `OptimalJInstallation\Solid_db` *folder. Copy the default* `solid.ini` *file to your Solid startup directory, for example* `OptimalJInstallation\SolidFolder`*, and modify it to accommodate your environment. If you configure Solid to use a port other than 1313, you must configure OptimalJ to use your selected port. See Configuring databases for more information.*

8. To start the SOLID server, double-click the *Solid for OptimalJ* shortcut on your desktop.

9. To stop the SOLID server, right-click the Solid FlowEngine program icon on the Microsoft Windows task bar and choose **Close** from the pop-up menu.

Step 2 - Create a new SOLID database (Linux)

1. In your file system, create a new folder for your SOLID database, for example:

   `OptimalJInstallation/SolidFolder`.

   Do *not* use the folder `OptimalJInstallation/Solid_db` because it contains the default CRM database.
2. Copy the SOLID license file `solid.lic` from `OptimalJInstallation/Solid4.0/eval_kit/standalone` to the new `OptimalJInstallation/SolidFolder` location.
3. The easiest way to create a new SOLID database is first to make your own copy of the file:

   `OptimalJInstallation/Solid4.0/standalone_eval_server_start`

   in the `OptimalJInstallation/SolidFolder` directory, calling the file something like `MySolid_server_start`.
4. Before you edit this file, change the file's permissions with the command:

```
chmod 711 MySolid_server_start
```

5. Edit the `MySolid_server_start` file with the command:

```
vi MySolid_server_start
```

6. Replace the path `eval_kit/standalone` by the path to your directory, for example `SolidFolder`. There are seven (7) occurrences that need to be replaced.
7. Locate the following lines in the `MySolid_server_start` file:

```
1 | # locate the executables directory
2 |   cd ./bin
3 |   binpath=`pwd`
4 |   cd ..
5 |   rootbytes=`pwd | wc -c`
6 |   bindir=`echo $binpath | cut -c $rootbytes- | cut -c 2-`
```

   Edit the relative paths in the two `cd` commands so the paths access the `OptimalJInstallation/Solid4.0/bin` directory from the `OptimalJInstallation` directory. The resulting paths are:

```
1 | # locate the executables directory
2 |   cd ./Solid4.0/bin
3 |   binpath=`pwd`
4 |   cd ../..
5 |   rootbytes=`pwd | wc -c`
6 |   bindir=`echo $binpath | cut -c $rootbytes- | cut -c 2-`
```

8. Save and exit your script (`Esc :wq`).
9. You create a new SOLID database by starting the server without an existing database. Execute your script from the `OptimalJInstallation` with the command:

```
SolidFolder/MySolid_server_start
```

Note: Because of the relative paths in the `MySolid_server_start` file, you must always start the database from the `OptimalJInstallation`.

10. As the folder `SolidFolder` is newly created and does not contain a `solid.db` file, the SOLID engine asks you if you want to create a new database.

    Answer YES to create a new database using the following properties:

Table 1-2 SOLID database properties

Property	Value
System catalog	optimalj
Username	optimalj
Password	optimalj

Figure 1-34 Creating a new database

Note: The string `optimalj` *is a default value which reflects the default settings within OptimalJ. If you wish to use different values for the above properties you would have to change the same properties within OptimalJ.*

Caution: Although the sample `standalone_eval_server_start` *and* `standalone_eval_server_stop` *scripts seem to indicate a port setting of 1315, Solid uses a default port of 1313.*

Note: If you are running another database concurrently using the same name on the same port, or if the port 1313 is already in use, you need to modify the `solid.ini` *file with appropriate name and the port settings. A default* `solid.ini` *file is located in the* `OptimalJInstallation\Solid4.0\eval_kit\standalone` *folder. Copy the default* `solid.ini` *file to your Solid startup directory, for example* `OptimalJInstallation\SolidFolder`*, and modify it to accommodate your environment. If you configure Solid to use a port other than 1313, you must configure OptimalJ to use your selected port. See Configuring databases for more information.*

11. To start the SOLID server at any time, use Step 2.9. As the `SolidFolder` is not empty the server automatically starts with your new database.
12. To provide a stop facility first copy the file:
    `OptimalJInstallation/Solid4.0/standalone_eval_server_stop`
    into the `SolidFolder` directory, calling the file something like `MySolid_server_stop`.
13. Edit the `MySolid_server_stop` file with the command:
    ```
    vi MySolid_server_stop
    ```
14. Replace the path `eval_kit/standalone` by the path to your directory, for example `SolidFolder`. There is one (1) occurrence that needs to be replaced.
15. Locate the following lines in the `MySolid_server_stop` file:
    ```
    1 | # locate the executables directory
    2 |   cd ./bin
    3 |   binpath=`pwd`
    4 |   cd ..
    5 |   rootbytes=`pwd | wc -c`
    6 |   bindir=`echo $binpath | cut -c $rootbytes- | cut -c 2-`
    ```
    Edit the relative paths in the two `cd` commands so the paths access the `OptimalJInstallation/Solid4.0/bin` directory from your `OptimalJInstallation` directory. The resulting paths are:
    ```
    1 | # locate the executables directory
    2 |   cd ./Solid4.0/bin
    3 |   binpath=`pwd`
    ```

```
4 |   cd ../..
5 |   rootbytes=`pwd | wc -c`
6 |   bindir=`echo $binpath | cut -c $rootbytes- | cut -c 2-`
```

16. Set the port number to shutdown. Replace the sample port number of `1315` with the default port number of `1313` or the port number you have set to match your environment. There are three (3) occurrences that need to be replaced.
17. Save and exit your script (`Esc :wq`).
18. To stop the SOLID server, execute your script from the `OptimalJInstallation` with the command:

```
SolidFolder/MySolid_server_stop
```

Note: Because of the relative paths in the `MySolid_server_stop` file, you must always stop the database from the `OptimalJInstallation`.

Step 3 - Confirm OptimalJ settings for SOLID

You can confirm the SOLID settings within OptimalJ by doing the following:

1. In OptimalJ, click **Tools>Options**on the menu to open the Options window.
2. Navigate to `OptimalJ Configuration.Code Generation.Databases` and select `Solid`.

Figure 1-35 Options for database configuration

Look at the *Database Driver* property. You can see that the JDBC driver needed for SOLID is set up when OptimalJ is installed.

The *User name* and *User password* properties are set to the default setting of `optimalj`.

Note: The User name and User password properties may be changed but then the Username and Password properties of SOLID must be changed to the same values. See Step 1.6.

3. Still in the Options window, select `OptimalJ Configuration.Testing.Database Configuration`.

Figure 1-36 Options for Testing Environment

4. Check that the *Deployment Database* is set to `Solid`.
5. Close the Options window.

You have created a new SOLID database and confirmed OptimalJ's settings to use it. You can now create and access tables in this database from OptimalJ by running scripts that are generated for an OptimalJ application.

If you need to recreate the CRM database tables and populate them, the file `crm.sql` is available in `OptimalJInstallation\Solid_db` to do so. You first need to drop the database tables using the SQL workbench, then you can use this file, by modifying your Solid shortcut (or script) with the extension: `-x execute:crm.sql`.

Note: Be careful when using table creation scripts. If you try to perform the same step-by-step instructions several times, you will likely attempt to create tables that already exist, which results in an error. If the tables already exist, you do not have to recreate them unless you want to start with an empty table. In this case, use the application's drop script to drop the tables, then the create script to recreate the tables.

1.4 Importing a domain class model

In this tutorial, you generate an OptimalJ application from a UML model. The model is delivered with OptimalJ, in the `OptimalJ installation directory\docs\tutorial` folder.

You import the model into OptimalJ with the Import facility for UML/XMI. The UML import facility of OptimalJ allows you to import models exported from:

- Rational Rose 2002
- Borland Together 6.1
- Enterprise Architect 3.51
- Objecteering 5.2.1

Also, import can be from a file containing a UML model. In this case, you must specify an additional XSLT file containing the transformation to be used.

Prerequisites

- You must be familiar with the basic development features of OptimalJ.
- You must be familiar with UML.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial you learn how to import a domain class model from a UML model.

Step 1 - Prepare the file system

In your file system, create the following directories:

- `\OptimalJ\importUML\umlModel`?this directory holds the model definitions.
- `\OptimalJ\importUML\umlEjbCode`?this directory contains the EJB application code.
- `\OptimalJ\importUML\umlWebCode`?this directory contains the Web application code.

Step 2 - Create a new project

1. On the menu, select **Project>New OptimalJ Project**. Enter `ImportUML` as the project name and click **Next**.
2. Select the type of project. Select **New Model** and set the **Model dir** to `\OptimalJ\importUML\umlModel`. Click **Next**.
3. Create a package structure. Enter `mycrm` in the **Fully-qualified Package Name** field. Set the initial package structure to `Three Tier Application Package` and click **Next**.
4. In the automount settings pane, select **Mount each filesystem yourself** and click **Next**.
5. Click **Finish**.
6. In the Explorer [Domain Model], expand `mycrm` to see the nodes in the domain package.

Step 3 - Import a UML model

You now import a UML model into the domain class package.

To import a UML model:

1. On the menu, choose **Model>Import Model>Import Domain Class from UML**.
2. Click the browse button and select the file `crm.xml` located in the `OptimalJ install directory\docs\tutorial` folder and click **Next**.

3. Select `mycrm.domain.class` as the target package and click **Next**.
4. Select **New import pattern** and enter `mycrmPattern` as the pattern name. Click **Next**.
5. Click **Finish**.
6. In the Explorer [Domain Model], expand the `mycrm.domain.class` node to check that the UML model is imported into the domain class model.

   Notice the `mycrmPattern` node that contains the mapping between the names of the elements in the UML file and the unique identifiers of the elements in the domain class model.
7. To display the class diagram for this domain, right-click `mycrm.domain.class`, and choose **Show Domain Class Diagram** from the pop-up menu. You can change the layout of the diagram by dragging the classes to new positions.

Figure 1-37 Domain class diagram

You imported a model that contains three classes and two associations. Notice that at this stage there is no domain unique constraint in the classes. You are now going to generate an application from this model.

Step 4 - Generate all application models

You can now generate all application models to create the DBMS, EJB and Web models.

To generate all models:

1. On the menu, choose **Model>Update All Models**.
2. Select `mycrm` and click **Finish**.
3. In the Explorer [Application Model], expand the node `mycrm.application.dbms.mycrm.Call` to see that an unique key `uniqueId` has been added as the primary key.

Step 5 - Generate and compile all code

When you have completed modeling, you generate and compile the code for the application.

To generate and compile the code:

1. On the menu, choose **Model>Generate All Code**.
2. When prompted for a directory for the Ejb code, click **Mount New Filesystem**.
3. Click **Local Directory** and click **Next**.
4. Browse to `\OptimalJ\importUML\umlEjbCode` and click **Finish**.
5. Click **OK** to start the Ejb code generation.

   Wait for the generation process to complete before continuing.
6. When prompted for a directory for the Web code, click **Mount New Filesystem**.
7. Click **Local Directory** and click **Next**.
8. Browse to `\OptimalJ\importUML\umlWebCode` and click **Finish**.
9. Click **OK** to start the Web code generation.

   Wait for the generation process to complete before continuing.
10. On the menu, choose **Project>Compile Project**.

Step 6 - Create the database tables

Because this tutorial is based on an import file in which the database structure is slightly different from the default CRM application, you need to create new database tables to test the application.

1. Start the SOLID server by double-clicking the Solid 4.0 icon on your desktop.
2. In the Explorer [Code Model], expand `umlModel.mycrm.application.dbms`.
3. Double-click `Solid_MetaMycrm.sqm`.
4. In the Connection window, click **OK**.
5. Click **Create** to load the SQL script for creating the database tables, then Exec Batch.
6. Close the SQL workbench.

Step 7 - Test the application

To test the application, you start the database server and application servers.

To test the application:

1. From the menu, select **Test>Start Application Server** to start the EJB and Web servers. Your CRM application starts automatically in your Web browser.
2. Proceed with testing the application.

In this tutorial, you generated an application from imported UML definitions containing multiple classes and attributes, as well as multiple associations. The UML import facility of OptimalJ allows you to import models created with Rational Rose 2002 or files created with the export to UML feature of OptimalJ. The maintenance pattern provides you with the functionality needed to retrieve, create, update and delete service agreements, customers, and calls, as well as the ability to associate a service agreement with a customer.

Further reading

For more information see also *Importing a class model* and *Elements common to UML and domain class model*.

1.5 Generating a domain model from database definitions

In this tutorial, you generate an application by importing database definitions. You use the CRM database which is made available as part of a OptimalJ installation. You import the definitions of this database in OptimalJ and use these definitions to generate a domain model. This approach simulates building an application starting from a legacy situation.

Prerequisites

- You must be familiar with the basic development features of OptimalJ, including the domain model.
- You must be familiar with databases.

Duration

This tutorial takes approximately one hour to complete.

Objectives

The focus of this tutorial is to read database definitions to generate an application. To simulate a legacy situation, you use the CRM database. You load a table definition from the database into an OptimalJ DBMS model via the JDBC driver, then you create a domain model from the DBMS and, finally generate and execute the application.

Step 1 - Prepare the filesystem

In your file system, create the following directories:

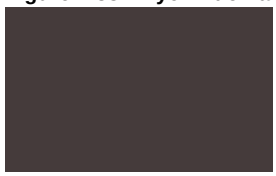
- `\OptimalJ\importDBMS\importDBMSModel`?this directory holds your model definitions.
- `\OptimlaJ\importDBMS\importDBMSEjbCode`?this directory contains the Ejb application code.
- `\OptimalJ\importDBMS\importDBMSWebCode`?this directory contains the Web application code.

Step 2 - Create a new project

1. From the menu, choose **Project>New OptimalJ Project**. Set the project name to `ImportDatabase` and click **Next**.

2. Select the type of project. Select **New Model** and set the **Model dir** to `\OptimalJ\importDBMS\importDBMSModel`. Click **Open** and then **Next**.
3. Create a package structure. Enter `mycrm` in the **Fully-qualified Package Name** field. Select `Three Tier Application Package` as the initial package structure and click **Next**.
4. Select **Mount each filesystem yourself** for the mount point settings and click **Next**.
5. Click **Finish**.

Figure 1-38 mycrm domain package



Step 3 - Import database definitions

In this step you import the CUSTOMER table of the CRM database schema.

1. Start the SOLID server by double-clicking the Solid 4.0 icon on your desktop.
2. From the menu, select **Model>Import Model>Import DBMS from JDBC Driver**.
3. Click **Test** to test the connection with the database. If the test is successful, the **Next** button becomes available.

Figure 1-39 Import DBMS from JDBC Driver: Connection properties

4. Click **Next**.

Note: If, after clicking the Test button, the Next button remains unavailable, verify the database URL (check the port number), driver (database specific), username and password (correct ones) and that the SOLID server is started.

5. Click **Deselect All**.
6. Select the CUSTOMER base table.

Figure 1-40 Import database definitions: select table

7. Once CRM.CUSTOMER is selected, click **Next**.
8. Select `mycrm.application.dbms` as the model package in which to import the database definitions.
9. Click **Finish**.

   In the output window you can see the progress. When finished the message 'Finished Importing Database Definitions' appears under the menu.
10. In the Explorer [Application Model], expand the nodes `mycrm.application.dbms` to check the imported definitions.

Figure 1-41 Application model generated from the CRM database schema

11. Check that the database model is correct. From the menu, select **Model>Check>Check DBMS Model**.

Step 4 - Generate the domain and application models

After the database definitions have been imported, you can generate the domain model from the DBMS model and then, from the domain model, generate the application models such as the EJB and WEB application models.

1. From the menu, select **Model>Generate Model>Generate Domain Models>Generate Domain from DBMS**.
2. Select `mycrm.application.dbms` and click **Next**.
3. Select `mycrm.domain.class` and click **Next**.
4. Leave all the model copier options unchecked and click **Finish** to generate the domain model from the database definitions.

   Wait until the message *Finished Incremental Generation of Domain from Database* appears under the menu.
5. Now, you need to generate the application model.

   From the menu, select **Model>Generate Model>Generate Application Models>Generate EJB from Domain** to generate the EJB model.
6. Select `mycrm.domain` and click **Next**.

   Select `mycrm.application.ejb` and click **Next**.

   Leave the model copier options unchecked and click **Finish**.

7. From the menu, select **Model>Generate Model>Generate Application Models>Generate WEB (EJB based) from Domain** to generate the Web model.
8. Select `mycrm.domain` and click **Next**.

   Select `mycrm.application.web` and click **Next**.

   Leave the model copier options unchecked and click **Finish**.

Step 5 - Generate and compile the code

After completing all modeling activities, you can generate and compile the code for the application components.

To generate and compile the code:

1. From the menu, click **Model>Generate All Code**.
2. When prompted for a directory in which to generate the Ejb code, click **Mount New Filesystem**.
3. Select **Local Directory** and click **Next**.
4. Browse to `\OptimalJ\importDBMS\importDBMSEjbCode` and click **Finish**.
5. Click **OK** to start the code generation.
6. When prompted for a directory in which to generate the Web code, click **Mount New Filesystem**.
7. Select **Local Directory** and click **Next**.
8. Browse to `\OptimalJ\importDBMS\importDBMSWebCode` and click **Finish**.
9. Click **OK** to start the code generation.
10. In the Explorer [Code Model], expand the nodes `mycrm.application.dbms` to check what has been generated.
11. From the menu, select **Project>Compile Project**.

    The message line in the main window shows the progress. It displays *Finished application* when compilation is complete.

Step 6 - Test your application

1. If the SOLID server is not running, start the SOLID server by double-clicking the Solid 4.0 icon on your desktop.
2. From the menu, select **Test>Start Application Server** to start the EJB and Web servers.

   Your application is automatically started in your Web browser.
3. Click **Maintenance Customer**.
4. Click **Browse**.

Figure 1-42 Accessing customer data

5. You can test the application by entering data or modifying and browsing through customer data.

In this tutorial, you used a database table to simulate a legacy situation. You imported the database definitions into the DBMS model. After that you generated the domain model from the DBMS model and then generated the other application models from the domain model. This reverse engineering approach allowed you to develop a three-tier application from a legacy situation. The result is a J2EE compliant application.

Further reading

For more information see also *Importing a domain model*.

1.6 Creating and distributing domain patterns

Domain patterns accelerate the process of creating a domain model by enabling you to capture and distribute domain models. In this tutorial, you create a domain pattern inside a domain pattern library that describes the Order and Orderline domain classes (defined as a composite association) a construct frequently used in business applications. You then create a domain pattern module to distribute this library. Finally, you install this domain pattern module in a new OptimalJ project and apply the domain pattern to a class model.

Prerequisites

You must be familiar with the basic development and testing features of OptimalJ. For more information, see the tutorial *Developing your first OptimalJ application*

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial you learn how to create, install and distribute domain patterns.

Step 1 - Create a new project

Create a new project and mount a directory.

To create a new project:

1. Create a new folder structure on a local file system called `OptimalJ\myDomainPattern`.
2. On the menu, select **Project>New OptimalJ Project**. Set the project name to `myDomainPattern` and click **Next**.
3. Set the type of project to `New Model` and click the browse button. Select `OptimalJ\myDomainPattern` and click **Open**. Click **Next**.
4. Accept the defaults for the fully-qualified package name and sub model name and set initial structure to `no initial structure`. Click **Next**.
5. Accept the defaults for the automount settings and click **Next**.
6. Click **Finish**.

Step 2 - Create a domain pattern library

Create a domain pattern library in which you define your domain pattern. A domain pattern library is a placeholder for domain patterns.

1. In the Explorer [Domain Model], right-click the Domain node and select **New DomainPatternLibrary**.
2. Enter `mypatterns` as the name for the library and click **Finish**. The domain pattern library is added directly under the Domain node.
3. Right-click the domain pattern library and select **New Child>ModelPackage**.
4. In the **Name** field, enter `orderorderline`, set the **Content Filter** to `domain` and click **Finish**.
5. Right-click the newly created (domain) model package and repeat the step 5 to create a model package with the name `class` and click **Finish**.
6. In the Properties window, set the *contentFilter* to `class`.
7. In the menu, select **Model>Import Model>Import Domain Class from UML**.
8. Click the browse button and browse to `OptimalJ installation directory\docs\tutorial` and select the file `ordordline.xml` and click **Open**, then **Next**.
9. Browse to `mypatterns.orderorderline.class` and click Next.
10. Select **New import pattern** and enter the name importOrderlinePattern. Click **Finish**.

    View the result in the Explorer [Domain Model]:

Figure 1-43 Creating a domain pattern library.

Note: You can also manually create domain patterns in the same way as creating a regular domain model, that is by manually adding classes, associations, and other domain model elements to the domain pattern.

Step 3 - Create a domain pattern module

The domain pattern module allows you to distribute your domain patterns to other developers in a project.

To create the domain pattern module:

1. In the Explorer [Domain Model], right-click the Domain node and select **New DomainPatternModule**.
2. In the **Name** field, enter `patterndistrib` and click **Finish**.
3. Select patterndistrib to display its properties in the Properties window.
4. Click the browse button of the *library* property to display the Property Editor. Click **Add** to add domain pattern libraries to the domain pattern module. Select mypatterns and click OK twice.

Note: Domain pattern modules can reference several domain pattern libraries.

5. In the Explorer [Code Model] open the `patterndistrib` package.
6. Right-click the `patterndistrib` folder and choose **Compile All** in the pop-up menu to create the installable domain pattern module:

Figure 1-44 Creating a domain pattern module

The `patterndistrib.jar` contains your installable domain pattern library available for distribution. The following steps demonstrate how you can install a domain pattern module.

Step 4 - Prepare the filesystem

Create a new directory structure to hold the model and code for a new application that will make use of the domain pattern you created in the previous steps. For example:

`\OptimalJ\domainPatterns` and under `\domainPatterns` create the subdirectories `\dpModel`, `dpEjbCode` and `\dpWebCode` to hold respectively your model definitions, EJB code, and Web code in separate subdirectories.

Step 5 - Create a new project

Create a new project, mount the directory you created above and create a default model package structure. (this step is required to simulate distributing the domain pattern to another developer.)

To create a new project:

1. On the menu, select **Project>New OptimalJ Project**. Set the project name to `DomainPatterns` and click Next.
2. Set the type of project to `New Model` and click the browse button and select the directory to mount `\OptimalJ\domainPatterns\dpModel` and click **Open**. Click **Next**.

3. Accept the defaults for the fully-qualified package name, sub model name and initial structure and click Next.
4. Accept the defaults for the automount settings and click **Next**.
5. Click **Finish**.

Step 6 - Install a domain pattern module

You can install a domain pattern module in OptimalJ making its pattern libraries available in your environment.

To install the domain pattern module:

1. On the menu, select **Tools>Options** to display the Options window. Browse to **IDE Configuration>System**. Right-click the Modules node and select **Add>Module**.
2. Browse to the local directory containing `patterndistrib.jar`, select this file and click Install to install the domain pattern module in your environment.

*Note: You can view the installed module in **IDE Configuration>System>Modules>Compuware Domain Pattern Modules**.*

3. Close the Options window. The result in the Explorer appears as shown in the example, and the message `Turning on modules...done` is displayed.

Figure 1-45 Installing a domain pattern module

Note: The JAR file is extracted to your user directory and mounted automatically by OptimalJ which makes the libraries available to the environment.

In this step, you installed a domain pattern module, in your environment allowing you to develop applications based on distributed domain patterns. This facility allows you to capture best practices in domain patterns and to distribute your models to other developers.

Step 7 - Create a domain model from domain pattern definitions

When you have installed a domain pattern module in your environment, you can use the domain patterns it contains to create new domain models. Or, you can apply domain patterns to existing domain models (weaving). This step describes the creation of a new domain model from a domain pattern.

To create a domain model from domain pattern definitions:

1. On the menu, choose **Model>Apply Domain Pattern**.
2. Select the pattern. Browse to `mypatterns.orderorderline.class` and click **Next**.
3. Select the target. Browse to `com.compuware.domainpatterns.domain.class` and click **Next**.

4. Click **Finish** in the wizard to create all the model elements contained in the domain pattern in the target class model.

*Note: The Apply Domain Pattern wizard provides options on how you can combine the domain pattern with the target (class model) by specifying the **Type of Weaving**. By default, the type of weaving is set to create for each model element (you can specify type of weaving for each model element):*

Figure 1-46 Applying a domain pattern

*Note: The **Copy onto** option is not available because the target model is currently empty, therefore you can only create or ignore elements (for each model element).*

5. View the result in the Explorer [Domain Model]:

Figure 1-47 Creating a domain model based on domain pattern definitions

In this step, you used an installed domain pattern module to create a new class model based on predefined definitions.

In the type of weaving panel of the Apply Domain Pattern wizard, the **Disconnected application** check box allows you to cut the link between the pattern and the model, with the consequence that if you update the pattern you cannot update the target. For more information, see the next step. The **Override regenerate properties** check box allows override the settings in the *regenerate* property in the target (the *regenerate* property for model elements determines the behavior of the incremental copier).

Step 8 - Update patterns

When you apply a pattern in connected mode (the **Disconnected application** is check box is *not* selected), you can modify the way domain patterns are combined with their targets.

To use the update pattern menu:

1. In the Explorer [Domain Model], expand the node `com.compuware.domainpatterns.domain.class`, right-click the weave root called class and select **Update Pattern** in the pop-up menu to display the Update Pattern wizard.

2. Expand the Salesorder node and select the attribute total. Select **Ignore** as the **Type of weaving**.

Figure 1-48 Update Pattern

3. Repeat step 2 for the attribute status. Click **Finish**.
4. Expand the class Salesorder to verify that the update pattern removed the two attributes total and status from the target model.
5. Optionally, generate the application models, generate and compile the code and test your application as described in steps 9 and 10.

Step 9 - Update all model and generate and compile the code

Update your application models with the definitions in your domain model. You can then generate code for the application models. Each model creates a number of files (Java, JSP, XML) from the model definitions.

1. Choose **Model>Update All Models**, select the root model package `com.compuware.domainpatterns`, and click **Finish**.
2. On the menu, select **Model>Generate All Code**.
3. When prompted, you need to mount the directories for the Web and EJB code:

   Select `OptimalJ\domainPatterns\dpEjbCode` and click **OK**.

   Select `\OptimalJ\domainPatterns\dpWebCode` and click **OK**.
4. On menu, choose **Project>Compile Project**.

Step 10 - Test the application

To test the application, you need to start the database and the application server.

To test the application:

1. Start the SOLID server by double-clicking the Solid 4.0 icon on your desktop. You have to run the database scripts as described in *Setting up a SOLID database.*
2. On the menu, select **Test>Start Application Server** to start the EJB and Web servers.
3. Proceed with testing the application as described in the tutorial *Your first OptimalJ application*.

In this tutorial, you created a domain pattern library to hold the definitions of your domain patterns. You then created a domain pattern from a UML/XMI import file (although you can create a domain pattern manually). You then created a domain pattern module with which you distributed your domain pattern. After installing the module in a new environment, you were able to develop a domain model from a pattern and to generate an application from it.

The **Update Pattern** menu provided you with the ability to modify the weaving of the pattern after you applied it.

Further reading

For more information on domain patterns, libraries, modules, and weaving, see the *Using OptimalJ* documentation.

1.7 Defining a domain service model

This tutorial shows you how to create a domain service model and how to generate the application model and code from this model. The service model allows you to define behavioral information. The main elements of the service model are domain services. Domain services contain service attributes, operations, and domain views. From the service model, you generate EJB session components in the EJB model and Web components in the Web model. At code level, this results in the

generation of session beans, JavaServer Pages, and other related code. Service operations become business methods on the session bean and Web actions. This tutorial uses the CRM example as the basis for which you develop a domain service. The domain service provides a domain view for ServiceAgreement, Customer, and Call.

Prerequisites

You must be familiar with the basic development features of OptimalJ.

Duration

This tutorial takes approximately a half-hour to complete.

Objectives

The focus of this tutorial is on creating a domain service that provides a domain view, and on generating the application model and the code based on the service model. For this purpose, you need a class model (imported from a UML/XMI file).

Step 1 ? Prepare the file system

Create a new directory for the example application; for example: `\OptimalJ\domainService`.

This directory will hold your model definitions, EJB, and Web code in separate subdirectories.

Step 2 ? Create a new project containing the CRM example

The CRM example is a sample application delivered with OptimalJ the demonstrates features and functionality available in OptimalJ. You can use the domain model of this application as a starting point for this tutorial.

To create a new project based on CRM:

1. From the menu, choose **Project>New OptimalJ Project**. Enter `domainService` as the **Name** and click **Next**.
2. Set the type of project to **Experiment with one or more example Models**. Set the **Unpack dir** to `\OptimalJ\domainService` (the directory you defined in Step 1) and click **Next**.
3. Select the **CRM Example (Sample domain model)** check box and click **Next**.

4. Click **Finish**.
   The result in the Explorer [Domain Model] appears as shown in the example.

Figure 1-49 Explorer window

5. Right-click the class node and select **Show Domain Class Diagram** to display the class diagram for this domain.

Figure 1-50 CRM class diagram

Note: Attributes and unique constraints are hidden on this diagram.

The class model contains three classes and two associations. Call is in a composite association with Customer, so call data is only accessible via Customer. The association between Customer and ServiceAgreement is mandatory?every Customer must have a ServiceAgreement.

The default application that can generated by this class model enables the end-user to display and maintain the data via two components?one for maintaining ServiceAgreements, and another for maintaining Customers and Calls. As a result of the composite association, the only way to create a Call is to create or edit a Customer. You cannot create a Customer without associating it with a ServiceAgreement. If no ServiceAgreements have been defined, it is not possible to create a Customer.

By defining a domain service, you can extend the basic functionality so that users can maintain both ServiceAgreements and Customers from one component.

Step 3 ? Create a domain service

You are going to create a domain service providing a view of ServiceAgreement, Customer, and Call. This domain service can then be used to generate the EJB and Web components required to maintain ServiceAgreement and Customer from one place.

1. In the Explorer [Domain Model], right-click the service model and choose **New Child>DomainService** to display the Create Domain Service wizard.
2. Set the **Domain Service Type** to **Define new View on Domain Class** and click **Next**.

*Note: The option **Do not Define View on Domain Class** is used to create purely behavioral domain services.*

3. Select `crm.domain.class.ServiceAgreement` as the root class on which to base the domain view. Click **Next**.
4. Do *not* change the default reference and click **Next**.

Figure 1-51 Setting the by value property for the reference view

The way a component uses an association is expressed as one of the following:

- `by value`?data of the associated class is included in the view
- `by reference`?only primary keys are included.

This setting enables the Customer data to be included in the view of Service Agreement root class. Call is also included because it is a composite part of an association with Customer, which is included in the view by value. Therefore, creating a view that includes Customer (by value), also automatically provides Call data.

5. Accept the suggested value `ServiceAgreementDomainView` in the **Name** field and click **Next**.

6. Enter `ServiceAgreementDomainService` in the **Name** field and click **Next**.
7. Do not create operations and click **Finish**.

   You have created a domain service containing a view of the class model. This view includes three classes (ServiceAgreement, Customer, and Call) with all their attributes. Expand the service node and then ServiceAgreementDomainService. The result in the Explorer [Domain Model] appears like this:

Figure 1-52 ServiceAgreementDomainService

It is possible to generate a service model from the class model, by selecting **Model>Generate Model>Generate Service from Class**. This scenario is not demonstrated in this tutorial.

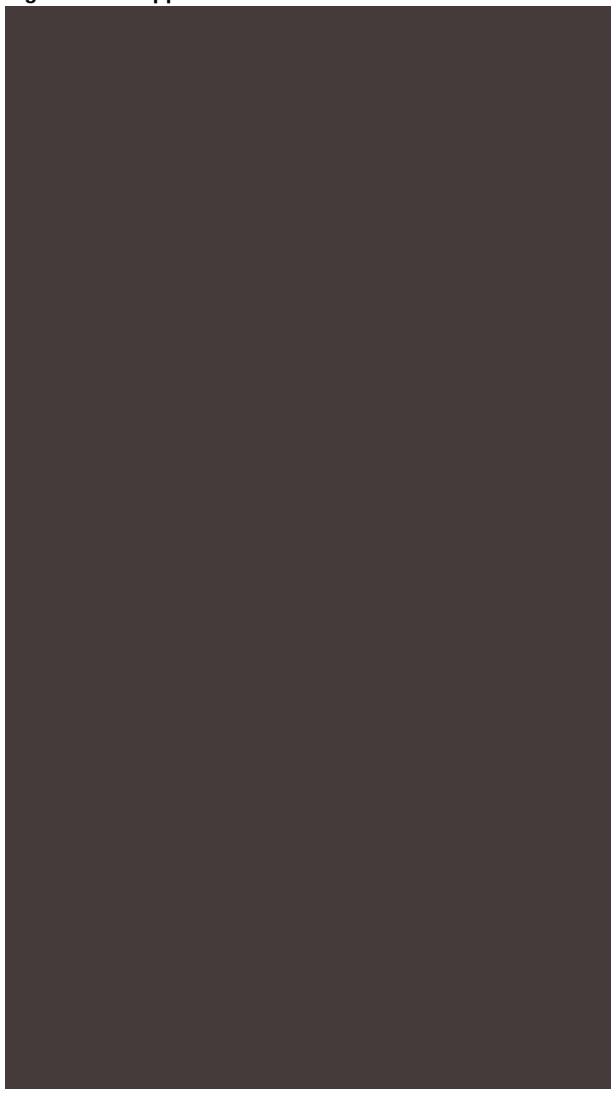
Step 4 ? Update application models

In this step you generate the Web, EJB and DBMS models from the domain model (the class and service models). The domain service definition is used to generate a session component in the EJB model and a Web component in the Web model.

1. From the menu, select **Model>Update All Models** to display the Update all models wizard.
2. Select `application` as the model package you want to update, and click **Finish**.

   The result in the Explorer [Application Model] looks like this:

Figure 1-53 Application models



By default, generating the application model results in EJB entity components that provide one view consisting of ServiceAgreement and Customer (by reference); the another view consisting of Customer, Call (by value), and ServiceAgreement (by reference).

As a result of defining a domain service, an EJB session component is also generated, which provides a view of ServiceAgreement and Customer (by value). It allows end-users to display and maintain the data of ServiceAgreement, Customer, and Call on one page.

Step 5 ? Generate and compile the code

Now you have completed the modeling, you must generate and compile the code for the application.

1. To generate the code, select from the menu **Model>Generate All Code**.
2. You are prompted to Select Filesystem to Generate code for Module: ejb. Select `\OptimalJ\domainService\crmEjbCode` and click **OK**.
3. You are prompted to Select Filesystem to Generate code for Module: Web. Choose `\OptimalJ\domainService\crmWebCode` to generate the code for the Web sources and click **OK**.
4. Wait for the generation process to complete, then choose **Project>Compile Project**.

   The message line in the main window shows the progress of the compiling process and reports *Finished project* when the compilation is completed.

Step 6 ? Create tables in the database

Before you can run the application, you need to create the database tables. The DBMS package contains a DBMS metafile and all the SQL scripts needed to drop, create, and initialize tables.

To create the table:

1. From the Start menu, start **SOLID for OptimalJ**, the default SOLID database.
2. In the Explorer [Code model], expand the node for the DBMS package.
3. Double-click the file `Solid_MetaCrm.sqm`.

Figure 1-54 Creating database tables

The OptimalJ SQL Workbench starts and shows the database connection window.

If you have properly configured your SOLID database settings (see *Setting up a SOLID database*), you can accept the default parameters.

4. Click **OK**.
5. In the SQL command window, click **Create** to load the Create SQL script in the command window.
6. Click **Exec Batch** to run the script.
7. Close the SQL Workbench.

Step 7 ? Test the application

To test the application, you need to start the Application Server and Web Server. When the Application Server starts, it generates stubs and skeletons for the Java beans. The EJB module deployment descriptors contain the information needed to generate and compile stubs and skeletons.

To start the EJB Server:

1. Choose **Test>Start Application Server**.

Additional tabs in the Output window show the progress of stub and skeleton generation, as well as information on the name server, the EJB Server, and Web server.

The Web server is also started, automatically executing the `MainMenu.jsp`, the home page of an OptimalJ Web application.

Figure 1-55 Main Menu

2. In your Web browser, click **Maintenance ServiceAgreementDomainService**.
3. To create a new service agreement, click **New** to display the form for maintaining service agreements and customers.
4. Enter data for the service agreement (the fields marked with an asterisk are mandatory).
5. To create a new customer for this service agreement, click **Create**.

6. Enter data for the customer.
7. To create a customer call for this customer, click **Create**.
8. Enter the call data and click **OK** to save the call information and return to the Customer page.
9. Click **OK** to save the customer information and return to the ServiceAgreement page.

Figure 1-56 Maintenance form for ServiceAgreementDomainService

10. Click **OK**.
11. Click **Submit** to store the data in the database.
12. Click **Browse**.

    You have created a service agreement, associate customer and call. The domain service made it possible to add the data to the database from a single component.

ServiceAgreementDomainService provides a view of ServiceAgreement and Customer data by values. You can compare this view to the default view available for ServiceAgreement. While the ServiceAgreement view relies only on the data supplied by the ServiceAgreement entity bean, the view provided here relies on the session bean ServiceAgreementDomainService that uses the ServiceAgreement and Customer entity beans to create this composite view.

In this tutorial, you used the CRM class model and extended it with a domain service that enables you to maintain ServiceAgreement and Customer. You created the domain service manually, but you can also generate automatically the service model from the class model.

Further reading

More information on domain services is available in *Using OptimalJ* documentation.

1.8 Defining a component model

The component model lets you visualize how OptimalJ components relate to one another. By default, OptimalJ components are related to one another by means of serving attributes (that is when you update the application model from the domain model definitions). However, you can also build the component model gradually using invocations. You can generate code to invoke multiple components methods by modifying the *usedComponent* and *usedOperation* properties of the client component. This is done in the Create Component wizard, in the Properties window, or by drawing dependencies between components in the Application Component Diagrams. The Service Locator, that is part of the Business Facade, takes care of processing invocations by locating the appropriate service objects through JNDI.

In this tutorial, we want to initialize the call severity attribute according to the service agreement service level attribute (all calls for a customer with gold service level being critical). To implement this requirement, you create a session component that invokes the findByProfileOnKey finder method on the ServiceAgreement entity component. The session component acts as a facade for the Web component Customer. The Web component invokes the session component to get the serviceLevel attribute and set the Call severity attribute based on the service level.

The following steps describe how to assign the session component with the Service Agreement entity component as *usedComponent* and the findByProfileOnKey finder method as *usedOperation*. Doing so allows you to invoke the finder method, retrieve a service agreement data via the Business Facade, and use this data to initialize the Call severity based on the service agreement level.

Prerequisites

- Familiarity with the basic development features of OptimalJ as described in the tutorial *Developing your first OptimalJ application*.
- Familiarity with the concepts of Web and session components.

Duration

This tutorial takes approximately one and a half hours to complete.

Objectives

This tutorial demonstrates how to use components and the operations they contain.

Step 1 - Prepare the filesystem

Create a new directory for the example application, for example:

- `\OptimalJ\CompModel`?this directory will hold your model definitions, and EJB and Web code in separate subdirectories.

Step 2 - Create a new project containing the CRM example

The CRM example is a sample application delivered with OptimalJ that demonstrates features and functionality available in OptimalJ. You can enable the CRM application when creating a new OptimalJ project.

To create a new project based on CRM:

1. On the menu, select **Project>New OptimalJ Project**. Set the project name to `ComponentModel` and click **Next**.
2. Select `Experiment with one or more example Models`, set the **Unpack dir** to `\OptimalJ\CompModel` and click **Next**.
3. Select the CRM Example (sample domain model) check box and click **Next**.
4. Click **Finish**.

Figure 1-57 CRM model

Step 2 - Create the Accessor domain service

You are now going to generate the domain service model from the class model and create an additional domain service called AccessorSvc. AccessorSvc acts as a facade to allow the Web component Customer to invoke the findByProfileOnKey method on the ServiceAgreement entity component.

1. In the service model package, right-click ServiceFromClassPattern and select **Update Model**. Two domain services are automatically generated.
2. Right-click the service model package and select **New Child>DomainService**.
3. Select **Do not use View on Domain Class**, and click **Next**.
4. In the **Name** field, enter `AccessorSvc` and click **Next**.
5. Create an operation `getSa` returning a String and taking saId (String) as parameter:

Figure 1-58 getSa operation

6. Click **Finish**.

Note: You do not need to create a domain service attribute.

7. On the menu, select **Model>Update All Models**, select `crm` and click **Finish**.

Step 3 - Modify the component model

You must now modify the component model and create dependencies between the components AccessorSvc, Customer, and ServiceAgreement.

1. In the Explorer [Application Model], right-click `ejb` and select **Show Application Component Diagram**.
2. In the Application Component Diagram, click the Dependency Edge tool (the arrow), and draw a dependency from AccessorSvc to ServiceAgreement.

Figure 1-59 EJB component model

3. In the Application Component Diagram, double-click AccessorSvc to open the Component Editor.
4. Click **Next** until you reach step 6 (Edit Used Components), click `crm.application.ejb.ServiceAgreement` to make it active and select the `crm.application.ejb.ServiceAgreement.findByProfileOnKey(DataTypes.ejbDataTypes.String)` check box.

Note: You have modeled your first invocation, and filled the usedComponent and usedOperation properties. From this definition, a method is generated in the code model to invoke the findByProfileOnKey method.

5. Click **Finish**.
6. Expand the node for the Web model package, right-click the Customer Web component and select **Edit**.
7. Click **Next** until you reach step 8 (Edit Used Components).

8. Click **Add**, select AccessorSvc and click **OK**.
9. Select the method `crm.application.ejb.AccessorSvc.getSa(DataTypes.ejbDataTypes.String)` check box.
10. Click **Finish**.

You have created two invocations. The First one by using the Application Component Diagram, the second one directly in the Component Editor. Note that although you could directly invoke the finder method from the Web component, it is generally considered a best practice to perform this access using a session component.

Step 4 - Generate code and use invocation methods

You have now created your component model, you need to generate code and use the invocation methods that are generated. To implement this tutorial's requirements, you need to add code in three places:

- In AccessorSvcBean?You must implement the getSa method. This method gets the service agreement id as parameter, it invokes the finder method which and calls the getServiceLevel method and returns this attribute to the calling method.
- In CustomerSelectAction?You must extract the service agreement id and use it as parameter to call your invocation method for getSa. This returns the service level that you make available to other Web actions by passing it to the session object.
- In CustomerCallNewAction?You must extract the service level from the session object and use it to initialize the call severity.

1. On the menu, select **Model>Generate All Code**.
2. You need to mount directories for the Web and EJB code. Select `\OptimalJ\CompModel\crmEjbCode` and click **OK**.
3. Select `\OptimalJ\CompModel\crmWebCode` and click **OK**.
4. Implement the getSa method. In the Explorer [Application Model], in the `ejb` model package, right-click the AccessorSvc session component and select **Edit Free Blocks in Generated Files>BusinessMethods>AccessorSvcBean.java>body (getSa)**.
5. In the Source Editor, copy or enter the following code inside the getSa method free block:

```
 1 |        try {
 2 |        Collection saColl =
invokeFindByProfileOnKeyServiceAgreement(saId);
```

```
 3 |         Iterator it = saColl.iterator();
 4 |         while (it.hasNext()) {
 5 |             ServiceAgreementLocal saUobj = (ServiceAgreementLocal)
   |               it.next();
 6 |             ServiceLevelEnum sen = saUobj.getServiceAgreement().
   |               getServiceLevel();
 7 |             returnValue = sen.toString();
 8 |             }
 9 |         } catch (Exception e) {
10 |             System.err.println("Exception occurred" + e);
11 |         }
```

Note: This methods returns a service agreement level.

6. Press F9 to compile the file. This also reports any potential error.
7. In the Explorer [Application Model], right-click the Web component Customer and select **Edit Generated Files>CustomerSelectAction.java**.
8. At the very bottom of the file, locate the free block before the finishAction method, and enter the following code in it:

```
 1 |
 2 |      ServiceAgreementKey myCustAttr = (ServiceAgreementKey)
   |        request.getAttribute("ServiceAgreementServiceAgreementKey");
 3 |      if(myCustAttr
   |        != null){
 4 |        try{
 5 |             String myCustString =
invokeGetSa(getAccessorSvcBusinessFacade(session),
   |                myCustAttr.getSaId());
 6 |             session.setAttribute("saLevel", myCustString);
 7 |             }
   |               catch(Exception e) {
 8 |             System.out.println("some exception occurred:
   |               " + e);
 9 |             }
```

```
10 |     }
```

Note: This method invokes the getSa method to get the service level.

9. Press F9 to compile the file. This also reports any potential error.
10. In the Explorer [Application Model], right-click the Web component Customer and select **Edit Generated Files>CustomerCallNewAction.java**.
11. At the bottom of the file, locate the free block after the `formBean.init(request);` and enter the following code in it:

```
 1 |
 2 |     String saLevel = (String) session.getAttribute("saLevel");
 3 |     if(saLevel.
   |       equals("GOLD")){
 4 |         formBean.setCallSeverity("CRITICAL");
 5 |     }
 6 |     else if(saLevel.equals("SILVER")) {
 7 |         formBean.setCallSeverity("NORMAL");
 8 |     }
 9 |     else {
10 |         formBean.setCallSeverity("ENHANCEMENT");
11 |     }
```

Note: Based on the service level you set the call severity.

12. Press F9 to compile the file. This also reports any potential error.

Step 5 - Compile and test the application

You now need to compile your code, start the default Solid database, start the application server, and test your application.

1. On the menu, select **Project>Compile Project**.
2. Start the SOLID server by double-clicking the Solid 4.0 icon on your desktop. This starts the OptimalJ default database that contains data for the CRM application.
3. On the menu select **Test>Start Application Server**.
4. To test the application, perform the following steps:

1. Click **Maintenance Customer**.

2. Click **Browse**.
3. Click **Edit** for the customer John Smith.

   Note that **saId** `0001` corresponds to a bronze service agreement level.
4. Click **Create** to create a new customer call.

   Note that the call severity is set to `enhancement`.
5. Enter some values and click **OK** to go back to the customer edit page.
6. In the customer edit page, click **Select** for customer **saId**, and select the gold service agreement level.
7. Click **Create** again and observe that the call severity is now initialized to `critical`.
8. Close your Web browser and stop the application server using **Test>Stop Application Server**.

The ability to define your component model and your method invocations allow for a greater flexibility when building or integrating an application. By using Application Component Diagrams or by editing components, you can model the way OptimalJ components interact with one-another. Note that not all invocation possibilities are supported, for more information refer to the *Further reading* section of this tutorial.

Further reading

To find more information on the component model and on invocations, see also *Component model concepts*.

1.9 Modifying Access Behavior

This tutorial shows you how to modify the access behavior to the data in a domain service. The service model allows you to define behavioral type of information, and also to pre-define the accessibility. The access behavior is modeled by using access properties. Access properties are boolean properties, defined at the domain services level. The transformation patterns use the access properties to determine what access is allowed to the different components, attributes and operations.

The modification of the access behavior results in removal of buttons, hyperlinks and/or fields. This tutorial uses the CRM example as the basis for which you generate a domain service. The domain service provides a domain view for ServiceAgreement, Customer, and Call, for which you define access behavior.

Prerequisites

- You are familiar with the creation a of domain service. See *Defining a domain service model* for the details.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial, you change the default access properties of elements from a domain service that is part of the CRM example application. By changing these properties, you learn how to modify the access behavior of the Web component generated from the domain service.

Step 1 - Prepare the file system

Create a new directory for the example application, for example: `\OptimalJ\AccessBehavior`.

This directory will hold your model definitions, EJB, and Web code in separate subdirectories.

Step 2 - Create a new project containing the CRM example

The CRM example is a sample application delivered with OptimalJ that demonstrates features and functionality available in OptimalJ. You can use the domain model of this application as a starting point for this tutorial.

To create a new project based on CRM:

1. From the menu, choose **Project>New OptimalJ Project**. Enter `AccessBehavior` as the **Name** and click **Next**.
2. Select **Experiment with one or more example Models**. Set the **Unpack dir** to the directory `\OptimalJ\AccessBehavior` you created in Step 1 and click **Next**.
3. Select the **CRM Example (Sample domain model)** Install check box and click **Next**.

4. Click **Finish**.

Step 3 - Generate the service model

In this step, you generate the service model from the class model. Using this option, OptimalJ by default generates a domain service for each domain class that is *not* composite part of a composite aggregation.

1. Choose **Model>Generate Model>Generate Domain Models>Generate Service from Class**.
2. In the Select Model Package that contains Class Model pane of the wizard, accept the default class package and click **Next**.
3. In the Select Model Package to generate Service Model in pane, accept the default service package and click **Next**.
4. Accept the defaults in the Select generation options pane and click **Finish**.

Figure 1-60 Domain model containing the generated service model

Step 4 - Modifying the access behavior of the domain service

You will change the functionality of the ServiceAgreementSvc domain service so that a user of the application can only read existing ServiceAgreements. The user cannot read or modify the ServiceAgreement attribute saPrice. The user can add Customers and Calls to a Customer. Customers cannot be removed. The user can add new Calls to a Customer but cannot update or read the Call attribute callResolved.

Modify the access behavior of the existing domain service:

1. In the Explorer [Domain Model], expand the `domain.service` package, select the serviceagreementsvc domain view.
2. In serviceagreementsvc, select the ServiceAgreement class view.
3. In the Properties window, set the *createAllowed* and the *deleteAllowed* to False.
4. Expand ServiceAgreement, choose saPrice, and in the Properties, set *readAllowed* and *updateAllowed* to False.
5. In serviceagreementsvc domain view, select the Customer class view, and in the Properties window set the *deleteAllowed* to False.
6. In serviceagreementsvc domain view, expand the Call class view, and choose callResolved.

Figure 1-61 Select callResolved in the Call class view

and in the Properties window set *readAllowed* and *updateAllowed* to False.

Step 5 - Update the application model

In this step you generate the Web, EJB and DBMS models from the domain model, that is the class and service models. From each domain service definition, a session component and a Web component are generated respectively in the EJB and Web models.

Note: Use this option also if you make changes in the domain model and propagate them to the application model.

1. From the menu, select **Model>Update All Models** to display the Update All Models wizard.
2. Select the crm model package and click **Finish**.
   The result in the Explorer [Application Model] looks like this:

Figure 1-62 Explorer [Application model] window

Step 6 - Generate and compile the code

You generate code for the application model. Each model creates a number of files (Java, JSP, XML) from the model definitions.

To generate the application code

1. In the menu bar, select **Model>Generate All Code**.
2. You need to mount directories for the Web and the EJB code.
   Select `\OptimalJ\AccessBehavior\crmEjbCode` and click **OK**.
   Select `\OptimalJ\AccessBehavior\crmWebCode` and click **OK**.
3. From the menu select **Project>Compile Project**.

Note: When the compilation has finished successfully, the message Finished Project AcccesBehavior appears in the Output Window [Compiler] window.

In the previous step, you generated the application code for the CRM example for which you created domain services. The domain service that you modified is implemented as a ServiceAgreementSvc EJB session component that serves a ServiceAgreementSvc Web component. In the Explorer [Application Model], double-click on a Web component in the crm.application.web package, to open the Web component Diagram. The diagram shows you how these components are related to the existing components of the CRM example application.

Figure 1-63 Overview of the components of the application

Step 7 - Test the application

To test the application, you need to start the database and the application server. You do not need to start the Web server. OptimalJ starts the Web server when the application server has started.

1. Start the start the SOLID server by double-clicking the SOLID 4.0 icon on your desktop.
2. From the menu, select **Test>Start Application Server** to start the EJB and the Web servers.
3. In your Web browser, click **Maintenance ServiceAgreementSvc**.

   Note that there is no hyperlink **New** to create a new ServiceAgreement.
4. Select **Browse**.

Figure 1-64 ServiceAgreementSvc

Note that saPrice is not visible

5. Click **Edit** to edit the first ServiceAgreement.
   Note that there is no button to delete the ServiceAgreement
6. In the list of Customers, click **Edit** to edit the first Customer
   Note that there is no button to delete the Customer.
7. Click **Create** to add a new Call to the Customer.

Figure 1-65 Adding a new Call to a Customer

Note that the field `callResolved` is not displayed.

8. Enter the following values

Table 1-3 Call attributes and values

Attribute	Value
callId	0004
callShortDescription	initial value missing
callDescription	not available

Accept the defaults for severity and timestamp. Click **OK**.

Figure 1-66 List of Calls after creation of a new Call

9. Click **OK** twice to finalize the update of the ServiceAgreement. The browser reports the message *ServiceAgreement successfully updated*.
10. Click **Submit** to commit the changes. The browser reports the message *ServiceAgreement successfully stored*.
11. Click **Home**.

In this tutorial, you learned how to modify the access behavior of a Web component generated from a domain service, to limit the access to create, read, update, and delete actions, and to prevent data from being displayed. You generated the domain services from the CRM example class model. The ServiceAgreementSvc domain service enables users to create and delete ServiceAgreements, and to create and delete Customers. You modified the access behavior of this domain service by using access properties. You disabled the removal and creation of ServiceAgreements, and the removal of Customers. You prevented ServiceAgreement and Call attributes from being updated or displayed.

Further reading

More information on domain services is available in *Using OptimalJ* documentation.

1.10 Adding business rules

Business rules specify functionality an application has to deliver, such as constraints on the values for a field, or dependencies between attributes or between classes.

Business rules can be expressed as (for example):

- Business expressions used within business methods
- Initial values for attributes

In this tutorial, you implement a business rule using a business method, a business expression, and initial values for attributes.

The business method calculates the average age of the calls in days for the CRM sample application, as follows:

```
averageAgeCall = timedifference(todaysDate and CallDate)
/ numberCalls;
```

A business expression defined in a business expression library performs the calculation of the time difference. The code for this business expression looks like:

```
timeDifference = (todaysDate.getTime() -
callDate.getTime()) / factorMilliSec;
```

Additionally, for the domain struct type `Address` and the domain fields `street`, `city`, `countryCode`, and `phone` you define initial values to populate these fields at runtime.

Prerequisites

- This tutorial uses the CRM database provided with the OptimalJ installation.
- You must be familiar with the basic development features of OptimalJ.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial, you learn how to define initial values for attributes, add a domain operation, create a business expression library, and define a business expression. You then implement a business method in a free block that invokes the business expression.

Step 1 - Prepare the filesystem

Create a new directory for the example application, for example:

- `\cpwr\oj\busrules`?this directory will hold your model definitions, and EJB and Web code in separate subdirectories.

Step 2 - Create a new project containing the CRM example

The CRM example is a sample application delivered with OptimalJ that demonstrates features and functionality available in OptimalJ. You can enable the CRM application when creating a new OptimalJ project.

To create a new project based on CRM:

1. On the menu, select **Project>New OptimalJ Project**. Set the project name to `AddBusinessRules` and click **Next**.
2. Select `Experiment with one or more example Models`, set the **Unpack dir** to `\cpwr\oj\busrules` and click **Next**.
3. Select the CRM Example (sample domain model) check box and click **Next**.
4. Click **Finish**.

Figure 1-67 CRM domain class model

Step 2 - Specify initial values for attributes

The *initialValue* property sets the value of an attribute when a new occurrence of a domain class is created.

To specify initial values:

1. In the Explorer [Domain Model], expand the domain model, and select the `domain.class.Address` struct type.

2. Expand `Address`, select the domain field `street`.
3. In the Properties window, click *initialValue* and then the browse button to open the Property Editor for initial values.

Figure 1-68 Initial Value browse button

Note: If the Properties window is not visible, open it by selecting ***View>Properties*** *in the menu bar.*

4. Set the language to `Constant`.
5. Enter `One Campus Martius` in the body of the initial value.
6. Click **OK**.
7. Repeat the steps 2 to 6 using the listed values:

Table 1-4 Initial values for Address fields

Domain field	initialValue Language	initialValue Body
city	`Constant`	`Detroit`
countryCode	`Constant`	`US`
phone	`Constant`	`+01`

Step 3 - Add a domain operation

The domain operation is used to create a business method in the EJB tier and a Web action to access this method in the Web tier. The business method causes the generation of a dedicated method in an entity bean.

1. In the Explorer [Domain model], right-click `domain.class.Customer` to display a context-sensitive menu.
2. Choose **New Child>DomainOperation** on the menu.
3. In the wizard, click **Add** to add the operation name. In the **Name** field, enter `averageAgeCall`.
4. Select `long` as *Return Type*.
5. Click **Add**, enter `resolved` as *Name*, choose `boolean` as *Type*, and `in` as *Kind*.

Figure 1-69 Create domain operation

6. Click **Finish**.
7. Update all models. Choose **Model>Update All Models** in the menu. Select `crm` and click **Finish** to update the application models.

   Wait for the generation process to complete.

Step 4 - Create a business expression library and a business expression

A business expression library holds reusable code that can, for example, be invoked from a business method.

To create a business expression library:

1. In the Explorer [Application Model], right-click `crm.application.ejb` and select **New Child>Business Expression Library** to display the Create BusinessExpressionLibrary wizard.
2. Enter `crmrules` in the **Name** field. Click **Next**.
3. Define an expression in the library. You use this expression to calculate the time (factored to days) from the difference between

two dates `toDate - fromDate` in the business method `averageAgeCall`. Click **Add** and enter `timeDifference` in the **Name** field. Select `long` for the return type.

4. Add two parameters using the listed values:

Table 1-5 parameters for timeDifference

Name	Type
`fromDate`	`Date`
`toDate`	`Date`

Figure 1-70 Create business expression wizard

5. Click **Finish**.

   The business expression library `crmrules` and the business expression `timeDifference` are added to the CRM application.

6. In the Explorer [Application Model], select `crm.application.ejb.crmrules.timeDifference`. In the Properties window, select the *body* property in which you can enter the Java code for the business expression. Click the browse button.

Remove the comment and enter the following Java code:

```
long factorMilliSec = 60000*60*24; // to make milliseconds
  a day
return(long)((toDate.getTime()-fromDate.getTime())/(factorMilliSec));
```

Caution: Make sure the language setting is Java.

7. Click **OK**.

Note: The code you enter here is generated in the body of a method in `Crmrules.java` *class. This method is invoked when using the library.*

Step 5 - Generate code

When the application model is ready, you can generate the application code.

To generate code:

1. On the menu, select **Model>Generate All Code**.
2. You need to mount directories for the Web and EJB code. Select `\cpwr\oj\busrules\crmEjbCode` and click **OK**.
3. Select `\cpwr\oj\busrules\crmWebCode` and click **OK**.

Step 6 - Implement the business method

The operation `averageAgeCall` was added to the entity bean `CustomerBean` in the form of a business method. In this step, you add the code that is executed in the body of the method.

1. In the Explorer [Application Model], expand the node `crm.application.ejb`.
2. Right-click the EJB entity component `Customer`, and choose **Edit Free Blocks in Generated files>BusinessMethods>CustomerBean.java>body(averageAgeCall)**.

Figure 1-71 Editing CustomerBean

The Source editor displays in blue the guarded blocks that cannot be modified by developers and, in white, free blocks in which you are invited to add your own code.

3. In the body of the business method, add the following code:

```
java.util.Date today = new java.util.Date(); //sets date
  of today
int counter=0;
for (Iterator i=callColl.iterator();i.hasNext();) {
  CallData callBrowse = (CallData) i.next(); //loop through
    Call of Customer
  if (resolved==callBrowse.getCallResolved()) { //if status
    as parameter
    counter++;
    returnValue = returnValue +
    Crmrules.timeDifference(callBrowse.getCallTimestamp(),
      today);
  }
}
if (counter == 0) // avoid division by zero
  returnValue = 0;
else
  returnValue =  (long)returnValue/counter; // average
```

Figure 1-72 Source Editor

4. Click **File>Save** to save the change made in this method.

Step 7- Compile the code

Now that all the code is generated and adjusted, you can compile it.

To compile the code:

1. On the menu select **Project>Compile Project**.
   The message line in the main window shows the progress and reports. It displays ?Finished crm? when the compilation has completed.

Step 8 - Start the database and the application server

Before running the application, OptimalJ compiles the stubs and skeletons for the Java beans and starts the name server and the EJB Server. The EJB module deployment descriptors contain the information to generate and compile stubs and skeletons. Then, OptimalJ runs the default Web server.

To start the database and the application server:

1. Start the SOLID server by double-clicking the Solid 4.0 icon on your desktop. This starts the OptimalJ default database that contains data for the CRM application.
2. On the menu, select **Test>Start Application Server** to start the EJB and Web servers.

Step 9 - Test the application

To test the application you need to execute the averageCall business method for a given customer, and create a new customer to experiment the *initialValue* property.

To test the application:

1. In your Web browser, click **Maintenance Customer** to display the Browse page.
2. Click **Browse**.
3. Click **Edit** for `0001 John Smith`.
4. Click **averageAgeCall** to call your business method.

Note: The hyperlink takes you to the screen where the callResolved parameter is entered.

5. Click **OK**.

   The operation you implemented as a business method proceeds to browse through the calls that have not been resolved to calculate the `averageAgeCall` and returns this value.

   Changing the input parameter you can also calculate `averageCallAge` for the calls which were resolved.

Figure 1-73 Result page

Note: Results vary depending on the input parameter (resolved) and the date of execution.

6. Now, create a new Customer.

Figure 1-74 Address with initial values

Note: The editing page contains values you defined for the Address struct type domain fields.

7. To stop the application, close the browser window.
8. On the menu, select **Test>Stop Application Server**.

In this tutorial, you enriched the CRM application with business rules. More can be done using OptimalJ, for example you can use business expressions for initial construction values, you can define delete rules for associations, you can define pre- and post- conditions for business methods, the options are there to use. Additionally, you can make your business expression dynamic by setting the *dynamic* property of the business expression to `True`. Dynamic business expressions are generated in XML files instead of Java files. You use the Business Expression Server to execute them.

Further reading

To learn more about business rules, business expressions and dynamic business expressions, see also *Business rules* in *Developing an application with OptimalJ*.

1.11 Creating a two-tier application (DAO component)

Data Access Object (DAO) components allow you to access data in a database directly from the Web tier. In this tutorial, you create data access objects (DAO) inside the DBMS model from an imported domain model. You then update the application model, generate code, and access the DAOs directly from the Web tier. This solution involves a page iterator for retrieving data using a paging mechanism. The data is sent page-by-page to the client, instead of as one large data set.

Prerequisites

You are familiar with the basic development features of OptimalJ and have followed the tutorial *Your first OptimalJ application*.

Duration

This tutorial takes approximately 35 minutes to complete.

Objectives

This tutorial teaches you how to create a two-tier application where the Web tier communicates with a database through a DAO component.

Step 1 - Prepare the filesystem

In your file system, create the following directories:

- `\OptimalJ\twotier\twotierModel`?this directory holds your model definitions and the DAO components.
- `\OptimalJ\twotier\twotierWebCode`?this directory contains the Web application code.

Step 2 - Create a new project

It is convenient to perform each tutorial in a new project.

To create a new project:

1. On the menu, select **Project>New OptimalJ Project**. Set the project name to `TwoTierApplication` and click **Next**.
2. Select the type of project. Select `New Model` and browse to `\OptimalJ\twotier\twotierModel`. Click **Open** and then **Next**.
3. Create a package structure. Enter `crmtwotier` in the **Fully-qualified Package Name** field.

   Set the initial package structure to `Two Tier Application Structure` and click **Next**.
4. Configure mount point settings. The code (EJB and Web) is generated in separate directories (or mount points). In this tutorial, the Web code requires a separate directory. Select `Mount each filesystem yourself`, click **Next**, and then **Finish**.

By selecting a two-tier initial package structure, you create a structure containing a domain model with a class model, and an application model with a database and a Web model. These models contain the appropriate Technology Pattern to generate elements for a two-tier application (DAO and Web components, and a database schema).

Step 3 - Import the CRM application

The CRM example is a sample application delivered with OptimalJ that demonstrates features and functionality available in OptimalJ.

The application is available as an XML import file (`crmtwotier.xml`) in `OptimalJ Installation directory\docs\tutorial`.

1. On the menu, select **Model>Import Model>Import Domain Class from UML**.
2. Browse to `OptimalJ Installation directory\docs\tutorial`, select the file `crmtwotier.xml`, click **Open**, then click **Next**.
3. Expand the nodes `Model.crmtwotier.domain`, select `class` as the model in which to import and click **Next**.
4. Do not modify the default import pattern, click **Finish**.

Figure 1-75 CRM domain class model

Step 4 - Generate the application model

DAO components communicate directly with the database. They are generated in the DBMS model from a class model.

To generate DAO and Web components:

1. In the menu bar, choose **Model>Update All Models**.
2. Select `crmtwotier` and click **Finish**.

Figure 1-76 Two-tier CRM application model

Web serving attributes point to serving attributes of DAO components (via the Web serving attribute's *usedServingAttribute* property). This allows DAO components to exchange data with Web components.

Step 5 - Generate and compile all code

After defining your application DBMS and Web models, you are ready to generate the application code (`.java`, `.sql`, `.properties`, `.xml`, and `.jsp` files) from the model definitions. When you have generated the code, you can modify the `dbms-dao.properties` configuration file for the page iterator. You then need to compile your code.

To generate and compile the code:

1. In the menu bar, select **Model>Generate All Code**.
2. When prompted for a directory in which to generate the Web code, click **Mount New Filesystem**.
3. Select Local Directory and click **Next**.
4. Browse to `\OptimalJ\twotier\twotierWebCode` and click **Finish**.
5. Click **OK** to start the code generation.
6. In the Explorer [Code Model], expand `\OptimalJ\twotier\twotierModel\crmtwotier\application\dbms`.
7. Double-click the `dbms.properties` file to verify that the database access properties correspond to those defined for the database you are using.

*Note: The database configuration is done in the **Tools>Options** menu.*

8. Double-click the `dbms-dao.properties` file to open it in the Source Editor.
9. Set the *pageSize* property value to 2, for service_agreement and customer:

Table 1-6 Setting the page size

Key	Value
`crmtwotier.application.dbms.service_agreement.pageSize`	2
`crmtwotier.application.dbms.customer.pageSize`	2

With this setting, each page containing data will display no more than two occurrences.

Note: The chapterSize value for DAO components using the page iterator is set by default to 400. The size of this property is limited by the memory available on your computer.

10. In the menu bar, choose **File>Save** to save the file.
11. On the menu, choose **Project>Compile Project** to compile the application code.

In this step you generated the code for your application model. From the definitions contained in the DBMS model, you generated Java code for the DAO components. DAO components communicate with the database via JDBC.

Step 6 - Create the database tables

Because this tutorial is based on an import file in which the database structure is slightly different from the default CRM application, you need to create new database tables to test the application.

1. Start the SOLID server by double-clicking the Solid 4.0 icon on your desktop.
2. In the Explorer [Code Model], expand `twotierModel.crmtwotier.application.dbms`.
3. Double-click `Solid_MetaCrmtwotier.sqm`.
4. In the Connect window, click **OK**.
5. Click **Create** to load the SQL script for creating the database tables, then Exec Batch.
6. Close the SQL workbench.

Step 7 - Test the application

To test the application you need to start the integrated Tomcat Web server.

To test the application:

1. On the menu, choose **Test>Start Application Server**
2. Test your application. In the main menu, click **Maintenance ServiceAgreement**. Click **New** to create a new service agreement. Repeat the operation to create at least 3 service agreements, click **Submit** when you are done creating service agreements.
3. Click **Browse**, then use the Prev and Next buttons to experiment with the page iterator.

Note: Try modifying the page size in the `dbms-dao.properties` *(restart the Web server every time you modify the properties file to reload the definitions).*

4. Stop the application server by choosing from the menu **Test>Stop Application Server**.

This two-tier application allows you to create and update records without requiring an EJB server. You tested DAOs through the Business Facade. The page iterator has the advantage of retrieving data in pages, which improves the performances when retrieving large data sets.

In this tutorial, you created a two-tier application in which DAO components access data in the database directly from the Web tier. You configured the page iterator in the `dbms-dao.properties` file. DAOs can work with or without the page iterator. You can enable or disable this functionality by setting the *supportPageIterator* property of DAO components. The page iterator provides a mechanism where data is retrieved in blocks, improving the performance of the application.

Further reading

For more information on DAO, see also DAO-related documentation topics, such as *DAO and EJB architectures comparison*.

1.12 Using the page iterator in a multitier environment

In this tutorial, you retrieve data using DAO components. For maintaining the data you use EJB entity components. DAO components are enhanced with the page iterator pattern to present pages of data, instead of a large data set. You retrieve data from the default SOLID CRM database.

Prerequisites

- You must be familiar with the basic development features of OptimalJ.
- This tutorial also assumes that you followed the tutorial *Your first OptimalJ application*.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial, the CRM sample application is extended with DAO components to enable a fast-lane reader pattern. In such an architecture, EJB session components use DAO components for reading to the database and EJB entity components for updates. DAO components, by default, use a page iterator to present the data in lists.

Step 1 - Prepare the filesystem

Create a new directory for the example application, for example:

- `\OptimalJ\daoThreetier`?this directory will hold your model definitions, and EJB and Web code in separate subdirectories.

Step 2 - Create a new project containing the CRM example

The CRM example is a sample application delivered with OptimalJ that demonstrates features and functionality available in OptimalJ. You can enable the CRM application when creating a new OptimalJ project.

To create a new project based on CRM:

1. On the menu, select **Project>New OptimalJ Project**. Set the project name to `PageIterate` and click **Next**.
2. Select the type of project, by setting the radio group to `Experiment with one or more example Models`, set the **Unpack dir** to `\OptimalJ\daoThreetier` and click **Next**.
3. Select the Install Example Module (Customer Relationship Management) check box and click **Next**.
4. Do not change the default, click **Finish**.

Figure 1-77 CRM domain class model

Step 3 - Generate the application model

To create the application models and the DAO:

1. On the menu, select **Model>Update All Models**.
   In the wizard, select `crm` as the top model to be updated and click **Finish**.
2. When the update is finished, on the menu, select **Model>Generate Model>Generate Application Models>Generate DAO from Domain**.
   In the wizard, select `crm.domain` and click **Next**.
   Select `crm.application.dbms`, click **Next** and then **Finish** to generate DAO data schemas, DAO components, and DAO key classes in the DBMS model.

Figure 1-78 CRM application model

3. Select the Customer DAO component.
   In the Properties window, note that the *supportPageIterator* property is set to `true`. Changing this property defines whether the component uses the page iterator functionality or not.

Step 4 - Update an EJB session component to use DAOs

The EJB session components act as a facade for DAO and EJB entity components. The page iterator is enabled in the DAO component. The EJB session component must be stateful to keep a reference of the list for the page iterator. Since EJB session components were generated when you updated the application model, you can modify these to use DAOs with page iterator.

1. In the Explorer [Application Model], expand `crm.application.ejb`.
2. Right-click CustomerSvc and select **Edit**.
3. Click **Next** until you reach step 4?Edit Availability / State in the wizard.
4. Set the **State of the component** to stateful.
5. Select the **Manage Reference to DAO component** check box.
6. Click **Next** twice and then **Finish**.
7. Double-click the `ejb` package to open the Diagram Editor.

Figure 1-79 CustomerSvc using DAOs

Note that the EJB session component CustomerSvc references both EJB entity and DAO components. It uses DAOs for fast-lane reading and EJBs for secure writing.

Step 5 - Generate and compile the code

You generate code for the application model. Each model creates a number of files (Java, JSP, XML) from the model definitions.

To generate the application code:

1. In the menu bar, select **Model>Generate All Code**.
2. You need to mount directory for the Web and EJB code.
   Select `\OptimalJ\daoThreetier\crmEjbCode` and click **OK**.
3. Select `\OptimalJ\daoThreetier\crmWebCode` and click **OK**.
4. On menu, choose **Project>Compile Project**

In this step, you generated the application code. In the DBMS model, you created the Java code that implements DAOs communicating to the database through JDBC. The CustomerSvc session component uses these DAO components for fast-lane reading.

Note: You can configure the data access information, and the page iterator settings by modifying the files `dbms.properties` and `dbms-dao.properties`.

Step 6 - Test the application

To test the application, you need to start the database and the application server.

To test the application:

1. Start the SOLID server by double-clicking the Solid 4.0 icon on your desktop. This starts the OptimalJ default database that contains data for the CRM application.
2. On the menu, select **Test>Start Application Server** to start the EJB and Web servers.
3. Proceed with testing the application as described in the tutorial *Your first OptimalJ application*.

The application behavior is similar to the standard CRM example provided, although here the communication with the database takes place both using DAO and EJB components. Additionally, you get the advantage of retrieving data in pages.

In this tutorial, you created DAO to access the data in the database, and you modified an EJB Session component to reference DAOs with page iterator.

The DAO can work with or without the page iterator, you learned how to set the *supportPageIterator* property of the DAO component.

The page iterator provides a mechanism where data is retrieved in blocks, improving performance when retrieving large data sets. This mechanism is used in combination with EJB entity components for secured updates of the database.

Further reading

For more information on the page iterator and DAO components, see also *Using DAO components in three-tier applications* and other related documentation topics.

1.13 Creating message-driven components

OptimalJ allows you to model an EJB message-driven component in the EJB model. This component represents a J2EE message-driven bean (a JMS message consumer). It can either consume all messages from a particular destination (topic or queue), or it can consume specific messages from the destination. Upon delivery of a JMS message, the EJB container instantiates the corresponding message bean that handles the messages. The JMS messages are generated by JMS message producers.

Prerequisites

- You must be familiar with the basic development features of OptimalJ.
- You should have a basic understanding of *Java Message Service (JMS).*

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial, you should understand the essential concepts and tasks for developing message-driven components. You create a JMS message and transport it via a JMS queue. You create a message-driven component, model a JMS message production, and test your application in OptimalJ.

Step 1 - Prepare the filesystem

Create a directory structure for the example application, for example:

- `\OptimalJ\mdComponent`?this is the root directory that will hold your model definitions, and EJB and Web code in separate subdirectories.
- `\OptimalJ\mdComponent\model`?this directory will hold your model definitions.
- `\OptimalJ\mdComponent\ejbCode`?this directory will hold your EJB code.
- `\OptimalJ\mdComponent\webCode`?this directory will hold your Web code.

Step 2 - Create a new project

You need to create a new project to run your tutorial.

To create a new project:

1. On the menu, select **Project>New OptimalJ Project**. Set the project name to `CreateMDC` and click **Next**.
2. Select the type of project, by setting the radio group to `New Model`, and set the **Model dir** to `\OptimalJ\mdComponent\model`. Click **Next**.
3. Type in the **Fully-qualified Package Name** `simple_jms` and click **Next**.
4. Accept the default automount settings and click **Next**.
5. In the next panel, accept the defaults and click **Finish**.

Step 3 - Create a JMS message

OptimalJ delivers functionality of defining JMS messages in the EJB model. The JMS messages are consumed by an EJB message-driven component.

When the JMS provider starts (in OptimalJ : the JBoss server), it creates JMS destinations. The JMS clients can then create and consume JMS messages from those destinations.

To generate a JMS message in the OptimalJ Explorer[Application model], follow the steps below:

1. In Explorer [Application Model] right-click the `simple_jms.application.ejb` node and select **New Child>JMSMessage**. This opens the Create JMS Message wizard.
2. Enter the name of the new JMS message. In the **Name** field enter `ExampleMessage`. Click **Next**.
3. Select the type of the JMS message as `Text Message` from the drop-down list. Click **Finish**.

Step 4 - Use EJB components to produce a JMS message

You use an EJB session component to publish messages to a JMS queue. Via an OptimalJ domain service you create a domain service operation. The domain service results in a session component with a business method. The session component is configured to produce a JMS message. As a result, OptimalJ generates a private method in the code which is responsible for the message production.

To create a JMS message producer, in OptimalJ, follow the steps below:

1. In the Explorer[Domain Model] window, right-click `service` node and select **New Child>DomainService**. This opens the Create domain service wizard.
2. Select `Do not use View on Domain Class` for the domain service type. Click **Next**.
3. Type the name of the domain service, for example, `SenderService`. Click **Next**.
4. Add a domain service operation with the name `sendMessage` and return type `Void`. Add a parameter of type String and call it `messageContent`. Accept the default `in` **Kind** for this parameter. Click **Finish**.
5. Choose **Model>Update All Models** from the OptimalJ menu and select the `application` package, as this is the package we want to have generated.
6. Check the level of the logged information for the EJB server. To allow the generation of logging trace information in the EJB server's console, select menu **Tools>Options**.
7. Go to **OptimalJ Configuration>Code Generation>Logging Code**.

8. Set the *EJB Tier* property to `Trace`. This creates entries in the source code, which are printed in the EJB server's console.

Step 5 - Create message-driven components

OptimalJ generates EJB message-driven components that encapsulate the functionality of a J2EE message-driven bean (MDB). The EJB message-driven component processes JMS messages, delivered by a JMS provider (JBoss server). The JMS messages are delivered to a JMS destination.

The following steps guide you to create a message-driven component:

1. In Explorer [Application Model], right-click the `simple_jms.application.ejb` node and select **New Child>EJBMessageDrivenComponent**.
2. Enter the name of the new component. In the **Name** field type `Receiver`. Click **Next**.
3. Enter the name and select the type of the JMS destination, according to the following values:

Table 1-7 Destination name and type

Field name	Value
Destination name	`ExampleQueue`
Destination type	`Queue`

*Note: Enter the destination name which will be used later as a JNDI name for JMS messages. Be aware that the **Destination name** is case-sensitive.*

Click **Next**.

4. Skip the Delivery Type panel and click **Next**.
5. Select the EJB module in which the EJB message-driven component is deployed. Select the current EJB module, `ejb`. Click **Next**.

6. Select the JMS messages consumed by the component. Select the `simple_jms.application.ejb.ExampleMessage` check box. Click **Finish**.

Note: If, in the dialog, no JMS message is chosen for consumption, the generated message-driven bean will consume all available messages declared in the model.

7. The new message-driven component, `Receiver`, appears in the Explorer[Application Model] tree. Expand the `simple_jms.application.ejb.Receiver` node.

   Observe the `JMSMessageConsumption` element and its properties. Notice that your EJB message-driven component consumes the JMS message, created in *Step 1 - Create a JMS message*.
8. Note the created elements in the Explorer[Application Model] window. For the `simple_jms.application.ejb` node, you should have a picture similar to the following:

Figure 1-80 Elements in the ejb node

Step 6 - Model the JMS message production

In this step, you have to model the generated session component to produce a JMS message. The EJB session component has a list of JMS messages that can be created from that component. This list is exposed via the *usedMessage* property that indicates which JMS messages are ?used? by this component. ?Used? means in this context that this component is responsible for the creation and sending of the messages. For each entry in the *usedMessage* collection, a private Java method is generated that has the following signature:

```
1 | private void produce<MessageName>(
2 |       String destinationName,
3 |       Class destinationType,
4 |       String usedConnectionFactory,
5 |       String payLoad)
6 | throws Exception;
```

This method is responsible for creating and sending a JMS message <MessageName> to JNDI destination `destinationName` using a `destinationType`. For more information, see *Message production and consumption* from the OptimalJ online help.

To model the JMS message production:

1. In Explorer[Application Model] right-click the `SenderService` session component and choose **Edit** from the pop-up menu. This starts the Edit Wizard. Click **Next** until you reach the Edit Produced Messages panel. Click **Add** and select the `ExampleMessage` element. Click **OK** to add the selected JMS message to the list of produced messages. Click **Next**.
2. Skip the next panel that lists JMS topics to produce messages to and press Next.
3. The next panel contains the JMS queues to which the JMS messages are to be sent. Add a queue by clicking the **Add**. In the field enter the JNDI name for a JMS queue, for example, `ExampleQueue`. Do not specify anything for the JMS queue connection factory. This way, OptimalJ is using the default connection factory of the integrated JMS provider?JBoss. Click **Finish**.
4. In Explorer[Application Model] select the root package `simple_jms` and generate all the code. Choose a proper mount point for the EJB and Web related code if asked.
5. Generate all the code. In the menu bar, select **Model>Generate All Code**.

6. You need to mount directory for the Web and EJB code.
   Via the wizard, select `\OptimalJ\mdComponent\crmEjbCode` and click **OK**.
7. For the Web tier, select `\OptimalJ\mdComponent\crmWebCode` and click **OK**.
8. Right-click the `SenderService` session component and choose **Edit Generated Files** from the pop-up menu. Select `SenderServiceBean.java`. Locate the method `sendMessage`. In the free block of the method add the invocation call to the private method `produceExampleMessage`. Add:

```
1 | try {
2 |    produceExampleMessage("ExampleQueue", Queue.class, "default",
  |       messageContent);
3 | } catch (Exception e) {
4 |    logger.error("Caught " + e.getClass().toString() + "
  |      while producing ExampleMessage, cause: "
5 |                + e.getMessage() );
6 | }
```

9. Compile your project. Select **Project>Compile Project** from the menu.

Since we did not provide a queue connection factory, in the previous step, we are allowed to pass '`default`' as the queue connection factory. In the EJB deployment descriptors a resource reference is added that maps 'jms/default' to the default connection factory for JBoss. This functionality is only available when deploying on JBoss, as other application servers not always provide a default connection factory. In this case, free blocks in the deployment descriptors are available where these mandatory references have to be added manually.

If the component has *usedMessages*, but you did not add any connection factories to its list, a resource reference is added to the deployment descriptor. The reference uses the default connection factory of JBoss called `ConnectionFactory`. This factory can be addressed by using the logical name `default` in the Java code (see the above code snippet). In addition, this functionality is only available when deploying on JBoss, as other application servers not always provide a default connection factory that is always present. In this case, the user is offered free blocks in the deployment descriptors where these mandatory references have to be added manually.

Although OptimalJ registers any non-existing topics or queues with JBoss, when added under **Tools>Options**, this is not the case for connection factories. Any additional connection factories have to be configured manually. See your JBoss manual for details. Once these custom connection factories exist, they can be referred to by the EJB component by adding the name of the factory to its *usedTopicConnectionFactory* or *usedQueueConnectionFactory* properties.

To be able to create custom connection factories, for example, `MyTopicConnectionFactory` as an actual name for the topic connection factory, the factory needs to be configured in JBoss. Currently OptimalJ does not offer any means to dynamically create connection factories. To workaround this, you can create a file `<...>-service.xml`, for example, `MyConnFactory-service.xml` with the following content:

Example: MyConnFactory-service.xml

```
 1 | <?xml version="1.0" encoding="UTF-8"?>
 2 | <!--
 3 |  - The 'custom' attribute of the MBean needs a unique name
   |    to make the OILServerILService
 4 |    unique and prevents conflicts with other service definitions.
 5 |  - The 'ConnectionFactoryJNDIRef' and 'XAConnectionFactoryJNDIRef'
   |    should get a unique
 6 |    name which will turn up in the jndi global name space.
 7 |  - The 'ServerBindPort' should be unique. When the port
   |    number equals another port,
 8 |    deployment of this file will fail with the message:
 9 |    ERROR
   |      [OILServerILService] Starting failed java.net.BindException:
   |      Address already in
10 |    use: JVM_Bind
11 |
12 |  Example:
13 |
14 |  <server>
15 |    <mbean code="org.jboss.mq.il.oil.OILServerILService"
16 |           name="jboss.
```

```
   |             mq:service=InvocationLayer,
17 |           type=OIL,
18 |           custom=MyService">
19 |      <depends optional-attribute-name="Invoker">jboss.
   |        mq:service=Invoker</depends>
20 |      <attribute name="ConnectionFactoryJNDIRef">MyExampleFactory</
attribute>
21 |      <attribute
   |        name="XAConnectionFactoryJNDIRef">MyExampleXAFactory</attribute>
22 |      <attribute
   |        name="ServerBindPort">1234</attribute>
23 |      <attribute name="PingPeriod">60000</attribute>
24 |      <attribute
   |        name="EnableTcpNoDelay">true</attribute>
25 |    </mbean>
26 |  </server>
27 |
28 |  The
   |    file should be copied to the
<userdir>\system\jboss\server\default\deploy
   |    folder
29 |  so that it will be deployed when the server starts. Note
   |    that the 'jboss' folder does
30 |  not exist until you start the EJB server for the first
   |    time..
31 |
32 | -->
33 | <server>
34 |   <mbean code="org.jboss.mq.il.oil.OILServerILService"
35 |          name="jboss.
   |            mq:service=InvocationLayer,
36 |          type=OIL,
37 |          custom=MyService">
38 |     <depends optional-attribute-name="Invoker">jboss.
   |       mq:service=Invoker</depends>
39 |     <attribute name="ConnectionFactoryJNDIRef">MyTopicConnectionFactory</
attribute>
```

```
40 |     <attribute
   |       name="XAConnectionFactoryJNDIRef">MyTopicXAConnectionFactory</
attribute>
41 |     <attribute
   |       name="ServerBindPort">1234</attribute>
42 |     <attribute name="PingPeriod">60000</attribute>
43 |     <attribute
   |       name="EnableTcpNoDelay">true</attribute>
44 |   </mbean>
45 | </server>
```

This file can then be copied to the `system\jboss\server\default\deploy` folder of your `userdir` so that it will be deployed when the server starts.

Step 7 - Test the JMS message consumption

Testing the JMS message consumption requires certain settings. The application server (JBoss) has to be aware of the JMS destinations it must support. The whole application starts via the Web. A hyperlink points to the method that produces the JMS message. The application server console shows the result of the message consumption.

To test JMS message consumption in OptimalJ:

1. Enter the JNDI name for the JMS queue. On the menu, choose **Tools>Options**.
2. Go to **OptimalJ Configuration>Testing>JMS>Queues**.
3. Right-click `Queues` node and select **Add queue**. A Queue dialog appears.
4. Enter the name of the queue to be used in your JMS application. For this tutorial, enter `ExampleQueue` in the dialog box. Click **OK**.
5. On menu, choose **Test>Start Logging Server** to trace the results from the application server.
6. On menu, choose **Test>Start Application Server**. This starts your application in the context of the application server.

Note: If you have not set up your internal test environment to start the browser automatically to display the MainMenu, enter `http://localhost:8081/MainMenu.jsp` *in your browser.*

7. In the Web browser, click the link **Maintenance SenderService**, and click **sendMessage**. Enter some text in the **messageContent** edit field, for example, `A simple text message` and click **OK**. You receive a notification about the delivery of the JMS message : `SenderServicesendMessage was successful`.
8. In OptimalJ, the Output Window, select the com.compuware.log4j.LogginSocketServer - I/O tab.
9. Note the results. The Output Window[com.compuware.log4j.LogginSocketServer - I/O] shows an output like this:

```
TRACE EjbTier (Receiver.java:84)-Received a TextMessage
  from destination ExampleQueue (queue), text is: A simple
  text message
```

   The JMS message is produced by your `SenderService` session component and is consumed by `Receiver` via the messaging service of JBoss.

In this tutorial, you used the JMS facilities provided by OptimalJ with the integrated JBoss application server. Note that executing this tutorial with another application server, requires some additional actions. The following optional step gives a general advice about how to deploy your application on other JMS providers.

Step Optional - Deploying the JMS application on a different application server than JBoss

In this step, you deploy the current tutorial example on BEA WebLogic Server (version 7), instead of JBoss. This requires manual configuration of the WebLogic application server (knowledge about this application server is assumed).

1. Prepare your file system as in *Step 1 - Prepare the filesystem*
2. Create a project as in *Step 2 - Create a new project*
3. Create a JMS message as in *Step 3 - Create a JMS message*.
4. Create an EJB message-driven component as in *Step 4 - Use EJB components to produce a JMS message*.
5. Create a message producer component as in *Step 5* - Create message-driven components.
6. Model a message production as in *Step 6 - Model the JMS message production* with the following change:

   In substep 3, you need to enter the name of the **QueueConnectionFactory**. As configuring WebLogic is not in

the scope of this document, this example is based on the `examples` server that ships with the WebLogic 7 application server. Based on this server you have the choice to use an existing connection factory or create your own. If you do not want to use the existing one, the shortest way is to clone one of the existing factories. Call the connection factory, for example, `MyQueueConnectionFactory`. This is the name you need to enter in your *usedQueueConnectionFactory* list.

7. Continue to model a message production as in *Step 6 - Model the JMS message production* with the following change:

   In substep 8, define the call to the generated private method:

```
1 | try {
2 |    produceExampleMessage("ExampleQueue", Queue.class,
"MyQueueConnectionFactory",
  |       messageContent);
3 | } catch (Exception e) {
4 |    logger.error("Caught " + e.getClass().toString() + "
  |      while producing ExampleMessage, cause: "
5 |                + e.getMessage() );
6 | }
```

Note: Since we provided a queue connection factory, in the previous substep, we need to pass `MyQueueConnectionFactory` as the queue connection factory. In the EJB deployment descriptors a resource reference is added that maps to the referred connection factory.

8. Continue with *Step 6 - Model the JMS message production*.
9. Mount the `applications` folder of the WebLogic server where this application is to be deployed.
10. Create an EAR file which is automatically deployed on the server.

Caution: This step assumes that the Weblogic application server is running and properly configured to run its JMS facilities. This includes adding the OptimalJ runtime libraries to the classpath of the server. The JMS queue names and queue connection factories must be also configured.

Right-click the mounted applications folder and select **New>All Templates**. Go to **Templates>Deploy>EarDef.eardef**. Click

Next. Enter the name of the eardef file, for example, `simple_jms`. Click **Finish**. This will start the OptimalJ Assembly Workbench. The eardef files contain definitions, required for the generation of the EAR files.

11. On the Modules tab, add the `ejb.jar` module and add the `web.war` module. With the `web.war` module selected, enter the context root at the bottom of the window, for example, `simple_jms`. For more information about creating an EAR file, see *Creating the application EAR*. Save the file and exit the Assembly Workbench by closing the window. Right-click the `simple_jms.eardef` file and select **Compile**.
12. The eardef file is compiled to an EAR file. The generated EAR file is deployed automatically in WebLogic server. After a successful deployment, navigate your Web browser to `http://<server_url>:7001/simple_jms/MainMenu.jsp`
13. Test your application.

In this tutorial, you learned the essential concepts and tasks for developing message-driven components. You created a JMS message and a message-driven component and modeled a JMS message production. At the end, you tested your application to generate JMS messages and consume them in OptimalJ.

Further reading

For more information, see the documentation on *Messaging*.

To learn more about the EJB message-driven component, refer to *EJB 2.0 specification*.

To learn more about the Java Messaging Service, refer to *Sun JMS tutorial*.

1.14 Creating JMS durable subscribers

Many enterprise applications cannot tolerate dropped or duplicate JMS messages and require that every message be received once and only once. The guaranteed messaging ensures that messages are faithfully delivered to their recipients. Mechanisms like message persistency and durable subscriptions are vital to guarantee the creation of a robust JMS application.

OptimalJ offers functionality to create durable JMS clients. While a durable subscriber is disconnected from the JMS provider, it is the responsibility of the server to store messages until the receiver is available or the messages expire. The durable subscribers are modeled with an associated client ID which is an unique identifier for the client. The JMS provider tracks the client ID to identify a durable subscription.

The current tutorial creates a message-driven component as a durable subscriber of a JMS message. A session component, situated in a separate EJB model, acts as a JMS message producer and sender. The two components are located and deployed in different packages to simulate a situation in which the message-driven component is unreachable causing the message to be retained by the JMS server until the subscriber is available again.

Prerequisites

- You should have a basic understanding of *Java Message Service (JMS)* .
- You should have completed the tutorial *Creating message-driven components*.

Duration

This tutorial takes approximately 1.5 hours to complete.

Objectives

At the end of this tutorial, you should understand the essential concepts and tasks for developing durable subscribers. You create a JMS message and transport it via a JMS topic. You create a message-driven component as a durable subscriber, model a JMS message production, and test your application in OptimalJ.

Step 1 ? Prepare the filesystem

Create a directory structure for the example application, for example:

- `\OptimalJ\mdDurable`?this is the root directory that will hold your model definitions, the EJB and Web code in separate subdirectories.
- `\OptimalJ\mdDurable\model`?this directory will hold the model definitions for the sender part of the application.

- \OptimalJ\mdDurable\ejbCode?this directory will hold the EJB code for the sender part of the application.
- \OptimalJ\mdDurable\webCode?this directory will hold the Web code for the sender part of the application.
- \OptimalJ\mdDurable\Receive?this directory will hold the model definitions for the receiver part of the application.
- \OptimalJ\mdDurable\ejbReceiveCode?this directory will hold the EJB code for the receiver part of the application.

Step 2 ? Create a new project

You need to create a new project to run your tutorial.

To create a new project:

1. On the menu, select **Project>New OptimalJ Project**. Set the project name to DurableMDC and click **Next**.
2. Select the type of project, by setting the radio group to New Model, and set the **Model dir** to \OptimalJ\mdDurable\model. Click **Next**.
3. Type in the **Fully-qualified Package Name** jms_sender and click **Next**.
4. Accept the default automount settings and click **Next**.
5. In the next panel, accept the defaults and click **Finish**.

Step 3 ? Create a JMS message

The JMS messages are modeled in the OptimalJ EJB model. When the JMS provider starts (in OptimalJ : the JBoss server), it creates JMS destinations. JMS clients can create and consume the JMS messages from those destinations.

To generate a JMS message in the OptimalJ Explorer [Application Model], follow the steps below:

1. In Explorer [Application Model] right-click the jms_sender.application.ejb node and select **New Child>JMSMessage**. This opens the Create JMS Message wizard.
2. Enter the name of the new JMS message. In the **Name** field type DurableTextMessage. Click **Next**.
3. Select the type of the JMS message as Text Message from the drop-down list. Click **Finish**.

Step 4 ? Use EJB components to produce a JMS message

You can use EJB components to generate messages and send them via JMS destinations to the messaging system. In this step, you create an OptimalJ domain service with a domain service operation. The domain service results in a session component with a business method. Use the EJB session component to publish messages to a JMS topic.

To create a JMS message producer:

1. In the Explorer[Domain Model], right-click service node and choose **New Child>DomainService**. This opens the Create domain service wizard.
2. Select `Do not use View on DomainClass` for the domain service type. Click **Next**.
3. Enter the name of the domain service, for example, `SenderService`. Click **Next**.
4. Add a domain service operation with the name `sendMessage` and return type `Void`. Add a parameter of type `String` and call it `messageContent`. Accept the default `in` **Kind** for this parameter. Click **Finish**.
5. Choose **Model>Update All Models** from the OptimalJ menu and select the `application` package, as this is the package we want to have generated.
6. Switch to the Explorer[Application Model] and expand the `jms_sender.application.ejb` node. Rename the EJB module from `ejb` to `ejbSender`. This prevents the JMS sender and the future JMS receiver JAR files from conflicting when both are deployed.
7. Check the level of the logged information for the EJB server. To allow the generation of logging trace information in the EJB server's console, select menu **Tools>Options**.
8. Go to **OptimalJ Configuration>Code Generation>Logging Code**.
9. Set the *EJB Tier* property to `Trace`. This creates entries in the source code, which are printed in the EJB server's console.

Step 5 ? Model the JMS message production

In this step, you have to model the generated session component to produce a JMS message. The EJB session component has a list of JMS messages that can be created from that component. This list is exposed via the *usedMessage* property that indicates which JMS messages are ?used? by this component. ?Used? means in this context that this component is responsible for the creation and sending of the messages. For each entry in the *usedMessage* collection, a private Java method is generated that has the following signature:

```
1 | private void produce<MessageName>(
2 |       String destinationName,
3 |       Class destinationType,
4 |       String usedConnectionFactory,
5 |       String payLoad)
6 | throws Exception;
```

This method is responsible for creating and sending a JMS message <MessageName> to JNDI destination `destinationName` using a `destinationType`. For more information, see *Message production and consumption* from the OptimalJ online help.

To model the JMS message production:

1. In Explorer[Application Model], right-click the `SenderService` session component and choose **Edit** from the pop-up menu. This starts the Edit Wizard. Click **Next** until you reach the Edit Produced Messages panel. Click **Add** and select the `DurableTextMessage` element. Press **OK** to add the selected JMS message to the list of produced messages. Click **Next**.
2. The panel Edit Topics To Produce To contains the JMS topics to which the JMS messages are to be sent. Add a topic. Click **Add**. In the **Topic** field enter the JNDI name for a JMS topic, for example, `DurableTopic`. Do not specify anything for the JMS topic connection factory. This way, OptimalJ is using the default connection factory of the integrated JMS provider?JBoss. Click **Finish**.
3. In Explorer[Application Model], right-click the root package `jms_sender` and choose **Generate Code** from the pop-up menu.
4. You need to mount directory for the Web and EJB code.

   Via the wizard, select `\OptimalJ\mdDurable\ejbCode` and click **OK**.
5. For the Web tier, select `\OptimalJ\mdDurable\webCode` and click **OK**.

6. Provide code for the message production invocation. Right-click the `SenderService` session component and choose **Edit Generated Files** from the pop-up menu. Select `SenderServiceBean.java`. Locate the business operation `sendMessage`. In the free block of the method, add the invocation call to the private method `produceExampleMessage`. Add:

```
1 | try {
2 |    produceDurableTextMessage("DurableTopic", Topic.class,
  |       "default", messageContent);
3 | } catch (Exception e) {
4 |    logger.error("Caught " + e.getClass().toString() + "
  |      while producing DurableTextMessage, cause: "
5 |                + e.getMessage() );
6 | }
```

Note: Since we did not provide a topic connection factory, in the previous step, we are allowed to pass '`default`' as the topic connection factory. In the EJB deployment descriptors a resource reference is added that maps 'jms/default' to the default connection factory for JBoss. This functionality is only available when deploying on JBoss, as other application servers not always provide a default connection factory. In this case, free blocks in the deployment descriptors are available where these mandatory references have to be added manually.

7. Build the code for the sender part of your application. In Explorer[Code Model], right-click the `\OptimalJ\mdDurable\model` folder and select **Build All**.

Step 6 ? Create a durable subscriber

The durable subscriber is modeled via the EJB message-driven component. The message-driven component is situated in a separate model. This is to allow the receiver part of the JMS application to be separated physically from the sender part, when deployed.

To create a durable subscriber:

1. In the Explorer[Code Model], mount a separate folder for the receiving part of this application. Select `\OptimalJ\mdDurable\Receive`.
2. In the Explorer[Domain Model], create a new root domain model package called `jms_receiver`. In the Create Model Package

wizard, choose a three-tier structure. For the **Filesystem** field, make sure you select the folder that was mounted in substep 1. Click **Finish**.

3. In Explorer[Application Model], right-click the `jms_receiver.application.ejb` model package and select **New Child>EJBModule**. Name the new EJB module `ejbReceiver`.
4. In Explorer [Application Model], right-click the `jms_receiver.application.ejb` node and select **New Child>EJBMessageDrivenComponent**.
5. Enter the name of the new component. In the **Name** field enter `Receiver`. Click **Next**.
6. Enter the name and select the type of the JMS destination. For the fields below, enter the following values:

Table 1-8 Destination name and type

Field name	Value
Destination name	`DurableTopic`
Destination type	`Topic`

*Note: Enter the destination name which will be used later as a JNDI name for JMS messages. Be aware that the **Destination name** is case-sensitive.*

Click **Next**.

7. Select `Durable` for the delivery type. This enables the **Client ID** field. Here you enter a unique identifier which the JMS Server uses to identify this message consumer. Type `ExampleClientId`. Click **Next**.
8. Select the EJB module in which the EJB message-driven component is deployed. Select the current EJB module, **ejbReceiver**. Click **Next**.

9. Select the JMS messages consumed by the component. Select the `jms_sender.application.ejb.DurableTextMessage` check box. Click **Finish**.

Note: If, in the dialog, you select no JMS messages for consumption, the generated message-driven bean will consume all available messages declared in the model.

10. In the Explorer[Application Model], select the package `jms_receiver`. Go to the properties and select *dependsOn*. Add the `jms_sender` package and click **OK**.

    Note the created elements in the Explorer[Application Model] window. You should have a picture similar to the following:

Figure 1-81 Elements in the Explorer[Application Model] window

11. Generate code for the receiver part of the application. Right-click the package `jms_receiver.application.ejb` and select **Generate Code**.

Via the wizard, select `\OptimalJ\mdDurable\ejbReceiveCode` and click **OK**.

12. Build the code for the receiver part of your application. In Explorer[Code Model], right-click the `\OptimalJ\mdDurable\Receive` folder and select **Build All**.

Step 7 ? Test the JMS message consumption

Testing the JMS message consumption requires certain settings. The application server (JBoss) has to be aware of the JMS destinations it must support. The whole application starts via the Web. A hyperlink points to the method that produces the JMS message. The EJB server console shows the result of the message consumption. In this step, the sender and the receiver part of the application are deployed separately. You make the receiver part unavailable for message delivery by undeploying it from the JMS server. Since the receiver part is developed as a durable subscriber, the JMS message will be preserved until the consumer is available again.

To test the JMS durable subscription in OptimalJ:

1. Enter the JNDI name for the JMS topic. On the menu, choose **Tools>Options**.
2. Go to **OptimalJ Configuration>Testing>JMS>Topics**.
3. Right-click the `Topics` node and select **Add Topic**. A Topic dialog appears.
4. Provide the name of the topic to be used in your JMS application. Enter `DurableTopic` in the dialog box. Click **OK**.
5. In the Explorer[Code Model], mount the `\system\jboss\server\default\deploy` folder. The folder is available under the folder that you selected to be your `userdir` in OptimalJ. This provides you with access to the folder in which all applications are deployed. The folder is needed to support *Hot Deployment* of the *receiving* part of our application in JBoss.

Note: The `\system\jboss\server\default\deploy` folder does not exist until you start the EJB server for the first time.

6. On the OptimalJ menu bar, choose **Test>Start Logging Server**.
7. To deploy the *sending* part of your application, select **Test>Start Application Server** on the OptimalJ menu. The Archive File Selector dialog opens and offers you available JAR and WAR files for deployment. Select the following files:

- \OptimalJ\mdDurable\model\jms_sender\application\ejb\ejbSender.jar
- \OptimalJ\mdDurable\model\jms_sender\application\web\web.war

Press **OK**. This action also starts the Web server.

8. Select the EJB file ejbReceiver.jar in the \OptimalJ\mdDurable\Receive\jms_receiver\application\ejb\ejbReceiver.jar folder. Right-click it and select **Copy**. Navigate to the mounted folder \system\jboss\server\default\deploy that was mounted above. Right-click this folder and choose **Paste>Copy** from the pop-up menu. This performs *Hot Deployment* of the ejbReceiver.jar, the output can be watched in the Output Window[EJB Server] tab. If the deployment it successful, the result in the output window should be similar to:

```
INFO  [EjbModule] Creating
INFO  [EjbModule] Deploying Receiver
INFO  [EjbModule] Created
```

9. In the Web browser, click Maintenance SenderService. Click sendMessage. Enter some text in the edit field, for example, A durable text message and press **OK**. You receive a notification about the delivery of the JMS message: SenderServicesendMessage was successful
10. In the OptimalJ Output Window, select the com.compuware.log4j.LogginSocketServer - I/O tab.
11. Note the results. The Output Window[com.compuware.log4j.LogginSocketServer - I/O] shows an output similar to:

```
TRACE  EjbTier (Receiver.java:84)-Received a TextMessage
  from destination DurableTopic (topic), text is: A durable
  text message
```

The JMS message is produced by your SenderService session component and is consumed by the Receiver via the messaging service of JBoss.

12. To see the effect of the durable subscription you make the receiver part of your application unavailable. In the Explorer[Code Model], navigate to the folder \system\jboss\server\default\deploy that was mounted above. Find the ejbReceiver.jar file and delete it. The Output

Window[EJB Server] tab reports the undeployment of the `ejbReceiver.jar` file:

```
INFO [EjbModule] Destroying
INFO [EjbModule] Remove JSR-77 EJB Module: jboss.management.
  single:J2EEApplication= ,J2EEServer=Single,j2eeType=EJBModule,
  name=ejbReceiver.jar
INFO [EjbModule] Destroyed
```

13. Go back to the Web browser that still points to the `SenderService` component. Again send a message `This is the actual durable message!`.
14. Repeat substep 8 and watch the Output Window[EJB Server] tab for output. It shows the messages being received directly after the deployment of the `ejbReceiver.jar` has finished. The output should be similar to:

```
TRACE EjbTier Receiver.java:84)-Received a TextMessage
  from destination DurableTopic (topic), text is: This is
  the actual durable message!
```

In this tutorial, you used the JMS facilities provided by OptimalJ with the integrated JBoss application server. Note that executing this tutorial with another application server requires some additional actions. See the optional step in *Creating message-driven components* tutorial, which gives a general advice about how to deploy durable subscribers on other JMS providers.

Step 8 ? Test JMS message persistence

The durability is only one side of the guaranteed message delivery. Along with durability, the JMS provider is responsible to offer a message persistence mechanism and a storage for sent messages. Message persistence is used when the JMS messages must not be lost in the event of a provider failure. By default, the message persistence is switched on. If a subscription is durable and the subscribers are not currently connected, then the JMS message is held by the message server until either the receiver is available, or the message expires. This also applies to non-persistent messages.

In this step, you send a message to the EJB server. Stop the server and restart it to observe the effect of the message persistence.

To test the message persistence

1. In the Explorer[Code Model], navigate to the folder `\system\jboss\server\default\deploy`. Find the `ejbReceiver.jar` file and delete it. The Output Window[EJB Server] tab reports the undeployment of the `ejbReceiver.jar` file.
2. In the Web browser, click `Maintenance SenderService`, and click `sendMessage`. Enter some text in the edit field, for example, `A text message to persist` and click **OK**.

   The JMS server (JBoss) stores the sent message in a reliable storage.
3. Stop JBoss server using **Test>Stop Application Server** menu in OptimalJ. This way you make the *sending* and the *receiving* part of the application unavailable.
4. Restart the application server by choosing **Test>Start Application Server** from the OptimalJ menu. The Archive File Selector dialog opens and offers you available JAR and WAR files for deployment. Select the following files:

- `\OptimalJ\mdDurable\model\jms_sender\application\ejb\ejbSender.jar`
- `\OptimalJ\mdDurable\model\jms_sender\application\web\web.war`

   Press **OK**. This action also starts the Web server.
5. In the Explorer[Code Model], select the EJB file `ejbReceiver.jar` in the `\OptimalJ\mdDurable\Receive\jms_receiver\application\ejb\ejbReceiver.jar` folder. Right-click it and select **Copy**. Navigate to the mounted folder `\system\jboss\server\default\deploy` that was mounted above. Right-click this folder and choose **Paste>Copy** from the pop-up menu. This performs *Hot Deployment* of the `ejbReceiver.jar`, the output can be watched in the Output Window[EJB Server] tab. If the deployment it successful, the result in the output window should be similar to:

```
INFO  [EjbModule] Creating
INFO  [EjbModule] Deploying Receiver
INFO  [EjbModule] Created
```

6. In the OptimalJ Output Window, select the com.compuware.log4j.LogginSocketServer - I/O tab.
7. Note the results. The Output Window[com.compuware.log4j.LogginSocketServer - I/O] shows an output similar to:

```
TRACE  EjbTier (Receiver.java:84)-Received a TextMessage
  from destination DurableTopic (topic), text is: A text
  message to persist
```

The JMS message is taken from the storage and delivered to the `Receiver` via the messaging service of application server.

In this tutorial, you learned the concepts and tasks for developing reliable message-driven application by creating durable subscribers. You created a JMS message and a message-driven component and modeled a JMS message production. At the end, you tested your application to generate JMS messages and consume them in OptimalJ.

Further reading

For more information, see the documentation on *Messaging*.

To learn more about the EJB message-driven component, refer to the EJB 2.0 specification, refer to *EJB specification*

To learn more about the Java Messaging Service, refer to *Sun JMS tutorial*.

1.15 Defining presentation model extensions

In OptimalJ, the model elements Web textarea type and Web input type are presentation model extensions allowing you to specify the format and appearance of Web attributes in JavaServer Pages. For example, you can set the date format or number format that is used to validate and parse user input, or you can select which HTML type is used to represent a Web attribute. You can also set different HTML attributes, including the CSS style and format strings.

Prerequisites

- Familiarity with the basic development features of OptimalJ as described in the tutorial *Developing your first OptimalJ application*.
- Familiarity with the concepts of Web components.

Duration

This tutorial takes approximately one hour to complete.

Objectives

This tutorial demonstrates how you can enhance the presentation in your OptimalJ Web applications without having to modify the generated JSPs.

Step 1 - Prepare the filesystem

Create three directories for the application, for example:

- `\OptimalJ\presentation\presentationModel`?this directory will hold your model definitions.
- `\OptimalJ\presentation\presentationWebCode`?this directory will hold the generated Web code.
- `\OptimalJ\presentation\presentationEJBCode`?this directory will hold the generated EJB code.

Step 2 - Create a new Project

To demonstrate the Web model extensions, you create a new project with an initial three-tier structure.

Create the new project:

1. From the menu, choose **Project>New OptimalJ Project**. Enter `Presentation` as the **Name** and click **Next**.
2. Set the type of project to **New Model**. Set the **Model dir** to `\OptimalJ\presentation\presentationModel` (the directory you defined in Step 1) and click **Next**.
3. Enter `presentation` in the **Fully-qualified Package Name** and select Initial Structure **Three Tier Application Structure**. Click **Next**.
4. Select **Mount each filesystem yourself** and click **Next**.
5. In the Include Source Code and Archives pane, accept the default and click **Finish**.

Step 3 - Create a Customer class

You first need to create a domain class that contains the attributes for which you want to define Web presentation types.

1. In the Explorer [Domain Model], expand `presentation.domain`.
2. Right-click the class package, and select **New Child>DomainClass**.
3. Enter `Customer` in the **Name** field and click **Next**.

4. Populate the Customer class with attributes as shown. Use the **Add** button to add a new attribute.

Figure 1-82 Create the Customer class

5. Click **Next**.

Note: This tutorial does not require that you define an operation or that you base this class on a supertype.

6. Click **Finish**.

Step 4 - Generate the application models

In this step you generate the Web model that is to be modified.

1. From the menu select **Model >Update All Models**.
2. Select the presentation package.
3. Click **Finish**.

At this point, if you generate the application code, including the Web code (JSPs and Web actions), you will generate the following JSP to create or edit customer information.

Figure 1-83 CustomerMaintChange.jsp

In this tutorial, you are going to implement the following presentation requirements:

- Contract date should be displayed using a full date format (Tuesday, April 12, 2003).
- Date of birth should be displayed using a custom date format.
- Notes should be displayed using a multiple line entry field.
- Password should be displayed as a password (****).
- LastName should be displayed in bold red text.

- CustomerId should be validated against a regular expression ([A-Z][A-Z][0-9][0-9][0-9][0-9])
- The fields should be reordered to provide a more usable interface (Notes should be last).

Note: In this tutorial, we are not using the DBMS code, as we only focus on demonstrating the possibilities offered by the Web model. The generated EJB code is only needed in the step where you create a regular expression. If you want to fully test the application, including data storage and retrieval, you must create the database tables from the database definitions, and start the database server before starting the application server.

Step 5 - Create a Web textarea type

The text input box for the domain attribute notes is too small and does not allow the user to scroll through the text easily.

To provide a larger field with vertical scroll bars, create a user-defined Web textarea type:

1. In the Explorer [Application Model], expand `presentation.application.web`, right-click the web model package and choose in the menu **New Child>WEBTextareaType**.
2. In the Create WEBTextareaType wizard, enter a **Name** for the text area `NotesTextArea`. Set the number of columns and rows to `6` and the number of columns to `60`. Click **Next**.

Figure 1-84 Web Textarea type wizard

3. Click **Next** You will not define a CSS Style or Class.
4. Apply the Web textarea type to a domain attribute. Click **Add**, select the domain attribute notes of Customer, click **OK**.

Note: Web input types and Web Textarea types can be shared among several domain attributes.

5. Click **Finish**.
6. From the menu, choose **Model>Update All Models**, select the presentation package and click **Finish**.

Note: As the Web textarea type wizard lets you apply your presentation type to attributes in the domain class model, you need to update the Web model to propagate the presentation type from the domain to the Web attributes.

Step 6 - Generate and compile the code

Each model creates a number of files (Java, JSP, XML) from the model definitions.

To generate the application code:

1. In the menu, select **Model>Generate All Code**.
2. You need to mount directories for the Web and the EJB code. Select **Mount New Filesystem** to assign the mount point for the EJB code.

   Select `\OptimalJ\presentation\presentationEJBCode` and click **OK**.
3. Select `\OptimalJ\presentation\presentationWebCode` and click **OK**.
4. Click **OK**.
5. Compile the code by selecting **Project>Compile Project**. The message *Finished Project Presentation* in the Output Window [Compiler] confirms the successful compilation of the project.

Step 7 - Test the application

To test the application, you need to start the Web server

To start the Web server:

1. In the Explorer[Code model], right-click `CustomerMaintChange.jsp` in `\OptimalJ\presentation\presentationWebCode` and from the menu, select **Execute**. The Web server starts and opens a browser with a data-entry form.

Figure 1-85 The resulting page

Note: There is a default HTMLTextarea type available for your attributes. For example, in the Explorer [Application Model], you can select the Web data attribute firstName and change the property webtype to `DataTypes.webDataTypes.HTMLTextarea`.

Note: The next time you are asked to test the application while the Web server is still running, select the menu option ***Execute (Force Reload)****. Using this option enables you to deploy your application without the need for stopping the Web server. OptimalJ does this for you.*

2. Close the Web browser window when you have finished testing.

Step 8 - Create a Web input type

For presentation purposes, the Customer's lastName attribute must appear in bold, red text. To achieve this, create a user-defined Web input type.

1. In the Explorer [Application Model], right-click the web model package and choose in the menu **New Child>WEBInputType**.
2. In the Create WEBInputType wizard, enter `RedText` in the **Name** field, select a **Text** input type and set the **Size** to `60`. Click **Next**.

Caution: The Size field is an integer that sets the display size of the input field. It does not influence the maximum input size validation.

Figure 1-86 Create Wizard

3. Specify the style using **CSS Style** `color:red; font-weight:bold` and click **Next**.
4. Select **none** for the data formatting type and click **Next**.
5. Apply the Web input type to a domain attribute. Click **Add**, select the domain attribute `lastName` of `Customer` and click **OK**. Click **Finish**.

6. In the web model package, right-click WEBEJBFromClassPattern and select **Update Model**.
7. Generate and compile the code.
8. Test the application.
   In the field **lastName**, enter the name `Smith`.

Figure 1-87 The resulting page

Note: To specify the value of the mark-up type, use single quotes. For example, `font-family: 'verdana'`*. Alternatively, use style classes defined in a CSS.*

9. Close the Web browser window when you have finished testing.

Step 9 - Define date formats

The user wants to see the Customer's contractDate in a full date format (*Tuesday, April 12, 1999*). To do this, you need to create another Web input type.

1. In the Explorer [Application Model], right-click the web model package and choose in the menu **New Child>WEBInputType**.
2. In the Create WEBInputType wizard, enter `MyDate` in the **Name** field, select the **Text** input type and click **Next**.
3. Click **Next**. You will not define a CSS Style or Class.
4. Select data formatting type **Date/Time** and click **Next**.
5. Select the date format **DateFormat.date FULL (e.g. Tuesday April 12, 1952)**. Click **Next**.
6. Apply the Web input type to a domain attribute. Click **Add**, select the domain attribute `contractDate` of `Customer` and click **OK**. Click **Finish**.
7. In the web model package, right-click `WEBEJBFromClassPattern` and select **Update Model**.
8. Generate and compile the code.
9. Test the application.

   Try to enter a contract date with a format that is different from the example given (for example 11-10-2003) and click **OK**. The application displays the message: *field Customer.contractDate : data does not conform to format set for this data type*. Next, enter a correctly formatted date (for example Monday, November 10, 2003) click **OK**.

Figure 1-88 DateFormat.FULL data entry

10. Close the Web browser window when you have finished testing.

Step 10 - Define date format patterns

Create a more complicated entry for the date fields in the presentation layer by defining your own format pattern. Create an Web input type to implement a *user-defined* date format.

1. In the Explorer [Application Model], right-click the web model package and choose in the menu **New Child>WEBInputType**.

2. In the Create WEBInputType wizard, enter `MyCustomDate` in the **Name** field, select the **Text** input type and click **Next**.
3. Click **Next**. You will not define a CSS Style or Class.
4. Select data formatting type **Date/Time** and click **Next**.
5. Select **SimpleDateFormat(uses formatstring)**. The field **Format String** becomes available. Enter your preferred date format, for example `yyyy MM d (EE) HH:mm`. Click **Next**.

Note: The SimpleDateFormat pattern represents the java.text.SimpleDateFormat class. SimpleDateFormat is a standard Java class for formatting and parsing dates in a locale-sensitive manner. For more information, refer to the Java 2 Platform API Specification.

6. Apply the Web input type to a domain attribute. Click **Add**, select the domain attribute `dateOfBirth` of `Customer` and click **OK**. Click **Finish**.
7. In the web model package, right-click `WEBEJBFromClassPattern` and select **Update Model**.
8. Generate and compile the code.
9. Test the application.

   Try to enter a date of birth with a format that is different from the example given (for example 10-03-1957) and click **OK**. The application displays the message: *field Customer.dateOfBirth : data does not conform to format set for this data type*. Next, enter a correctly formatted date of birth (for example 1957 10 3 (Thu) 10:30) and click **OK**.

Figure 1-89 SimpleDateFormat(uses formatstring)

10. Close the Web browser window when you have finished testing.

Step 11 - Define password fields

Password fields in Web pages contain sensitive information. Password field are normally presented as normal text. Create a Web input type to implement a password field to hide the information entered in the Customer's password field.

1. In the Explorer [Application Model], right-click the web model package and choose in the menu **New Child>WEBInputType**.
2. In the Create WEBInputType wizard, enter `MySecretWord` in the **Name** field, select the **password** input type and click **Next**.
3. Click **Next**.
4. Apply the Web input type to a domain attribute. Click **Add**, select the domain attribute password of Customer and click **OK**. Click **Finish**.
5. In the web model package, right-click `WEBEJBFromClassPattern` and select **Update Model**.
6. Generate and compile the code.
7. Test the application.

   Enter text in the **password** field. Verify that every character you type in the field is displayed as a * .
8. Close the Web browser window when you have finished testing.

Step 12 - Validate input using regular expressions

Certain attributes must adhere to a fixed format, such as ZIP codes or car registration numbers. The user input is checked against this format to prevent an invalid entry in the JSP. To perform the check, you create a regular expression defined within a business expression. The regular expression is attached to the attribute using a domain attribute constraint. In this tutorial, the customerId field must conform to the format of two capital letters followed by four numbers, for example, VN3161.

1. In the Explorer [Application Model], right-click the ejb model package and select **New Child>BusinessExpressionLibrary**.
2. Enter `myexpressionlib` in the **Name** field and click **Next**.
3. Add a business expression to the business expression library as shown:

Figure 1-90 Create BusinessExpression Wizard

Click **Finish**.

4. In the Explorer [Application Model], select the `validCustomerId` business expression you just created. In the Properties Window, click the *body* property browse button to display the Property Editor. Select Language

RegularExpression and in the Body field you remove the comment and enter *^[A-Z][A-Z][0-9][0-9][0-9][0-9]$*.

Note: The caret (^) sign marks the start of the expression string, and the dollar ($) sign the end of the string. Regular expressions consist of digits [0-9], uppercase characters [A-Z], lowercase characters [a-z], combinations [A-Za-z].

5. Click **OK**.
6. In the Explorer [Domain Model], right-click the domain class `Customer` and choose in the menu **New Child>DomainAttributeConstraint**. In the wizard, enter the **Name** `checkId` and click **Next**.
7. Click **Next**. You will not define a constraint range.
8. Click **Add**, this automatically selects the business expression you have created. Click **Finish**.
9. Select the customerId domain attribute, and in the Properties window change the *attributeConstraint* property so that it references the checkId domain attribute constraint.
10. On the menu, choose **Model >Update All Models**, select presentation and click **Finish**.
11. In the Explorer [Application Model], expand the node `presentation.application.ejb` and select the ejb module.
12. In the Properties, select the *containedFiles* and add `/presentation/Myexpressionlib.class`.

Note: The class file for the business expression library starts with uppercase so enter the name accordingly.

13. Generate and compile the code.
14. Test the application. From the menu, select **Test>Start Application Server**.

    Enter a value in the **customerId** field that does not comply to the format as described in the body of the business expression. The application displays the message *field Customer.customerId : data is out of range*. Verify that an expression like **VN3161** is accepted.
15. Close the Web browser window when you have finished testing.

Step 13 - Reorder the fields

Reordering the fields can be done in the domain model, in which case the order is propagated to the other models, or it can be done in the application models (for example the Web model).

1. In the Explorer [Application Model], expand the nodes `presentation.application.web.customer`, right-click the `Customer` Web data class and select **Properties**.
2. In the Properties window, click inside the *feature* property field and then click **Browse**.
3. The feature Property Editor lets you reorder the attributes. Select notes and click **Down** until the notes attribute appears last in the list.
4. Use the Up and Down buttons to reorder the attributes according to the following list:

- customerId
- firstName
- lastName
- dateOfBirth
- contractDate
- password
- notes

5. Click **OK** and close the Properties window.
6. Generate and compile the code.
7. Test the application.

Figure 1-91 Changed order of input fields

8. Close the Web browser window when you have finished testing.
9. Stop the application server by selecting **Test>Stop Application Server** from the menu. OptimalJ stops the application server and the Web server.

Web presentation types allow you to specify the format and appearance of attributes in the HTML generated by the JavaServer Pages. You can set the date format or number format that is used to validate and parse user input. You can select which HTML type is used to represent an attribute. They also set some HTML attributes, including the CSS style. The maximum length of an attribute can be specified in the domain, EJB and Web model. OptimalJ supports reuse of presentation types and consistency of the presentation types that are used for Web data attributes that are based on the same model domain attribute.

Note: For more information, refer to the Java 2 Platform API Specification: DateFormat, SimpleDateFormat, NumberFormat, DecimalFormat, Jakarta Regular Expression.

Further reading

For more information on customizing the Web user interface, see also *Defining Web input types* and other related documentation topics.

1.16 Integrating with CORBA

This tutorial shows you how to integrate a CORBA IDL definition into your OptimalJ environment, using a simple CORBA implementation. An IDL file called `account.idl` is provided, which describes the interfaces of an account component that can be used to deposit and withdraw amounts in an account.

Prerequisites

Before starting this tutorial, you must have Sun JDK 1.4 installed.

Duration

This tutorial takes approximately one hour to complete.

Objectives

After completing this tutorial, you should understand how to integrate a CORBA component in your OptimalJ application.

Step 1 ? Prepare the filesystem

During the course of this tutorial, you define models and generate EJB and Web code. This information should be stored in separate directories and mounted for your project.

1. Create a new directory for the example; for example: `\OptimalJ\corbaIntegration`.
2. In this directory, create three subdirectories:

- `\OptimalJ\corbaIntegration\corbaModel`
- `\OptimalJ\corbaIntegration\corbaEjbCode`
- `\OptimalJ\corbaIntegration\corbaWebCode`

Step 2 ? Create a project

This tutorial requires that you create a model package with the name `account`.

To create a new project:

1. Choose **Project>New OptimalJ Project**. Set the project name to `corbaIntegration` and click **Next**.
2. Select **New Model** as the project type and specify the directory `\OptimalJ\corbaIntegration\corbaModel` in the **Model dir** field.

   Click **Next**.
3. Enter `account` as the **Fully-qualified package name** and choose `Three Tier Application Structure with Integration` as the **Initial Structure**. Click **Next**.

Note: You MUST use these values. The CORBA sample used for this tutorial depends on this package structure.

4. Choose **Mount each filesystem yourself** and click **Next**.
5. Ensure that all check boxes are cleared and click **Finish**. This creates packages for the domain, application, and integration models. Your start environment should look like this:

Figure 1-92 Start environment

6. Choose **File>Mount Filesystem**. In the wizard, select **Local Directory**, and select `\OptimalJ\corbaIntegration\corbaEjbCode`. Click **Finish**.
7. Repeat substep 3. to mount the Web code directory `\OptimalJ\corbaIntegration\corbaWebCode`.

Step 3 ? Import an IDL file

In the course of importing the CORBA IDL file, you create a CORBA client module.

To import the sample IDL file:

1. Choose **Model>Import Model>Import CORBA IDL from File** to start the import wizard.
2. In the **IDL Filename** field, select the file `UserDirectory\sampleprojects\integration\corba\simpleconnector\account.idl`.

   This directory is located in the user directory. On Windows, this is `C:\Documents and Settings\User\.OptimalJ-Edition\3.1` by default; on UNIX, it is `$HOME`.

   The fields **Additional include directories** and **Additional defines** are optional. Leave these three fields blank and click **Next**.

   (The import facility reads the IDL file from the file system and deals with preprocessing directives such as `#includes` and `#defines`. If you need this for importing your own IDL file, enter a semicolon-separated list of `#include` directories and a semicolon-separated list of definitions in the respective fields.)
3. Select `account.integration.corba` as the CORBA package for import and click **Next**.

4. Click **Create New**. This starts a separate wizard for creating a new CORBA client module.
5. Select `account.integration.corba` and click **Next**.
6. Accept the default name (`CORBAClientModule`) and click **Finish**.
7. Select the `CORBAClientModule` and click **Finish**.

   Your tree now looks like this:

Figure 1-93 After importing the IDL file

Step 4 ? Generate domain and application models

By generating an EJB model based on the integration model, you can generate a session bean (and related code) to serve as a wrapper for the connector client code. Although you can generate the EJB model directly from the integration model, it is better to first generate a domain model. This enables you to generate a Web model that can be used to create a user interface for the CORBA component.

1. Choose **Model>Generate Model>Generate Domain Models >Generate Domain from Corba** to start the wizard.
2. Select `acount.integration.corba` as the Corba package and click **Next**.
3. Select **account.domain** as the domain model and click **Finish**.
4. Choose **Model>Update Models**.
5. Select `account.application` as the top-level package to update and click **Finish**.

Step 5 ? Generate and compile code

After generating the EJB and Web models, you can generate code for the application and integration models.

To generate the code:

1. Choose **Model>Generate All Code**
2. When prompted for the location of generated code modules, choose `\OptimalJ\corbaIntegration\corbaEjbCode` for the EJB module code and `\OptimalJ\corbaIntegration\corbaWebCode` for the Web module code.
3. To view the structure of the generated code, use the Explorer [Code Model].

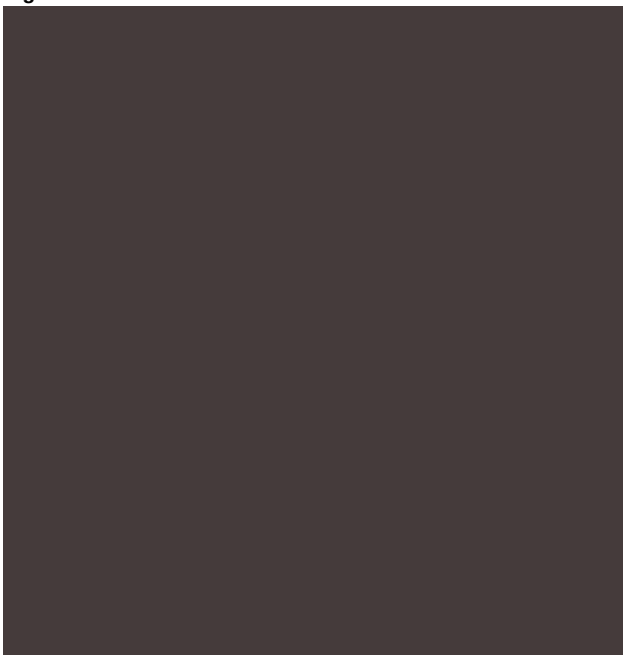
   For example, in the directory containing the integration models and code, navigate to `account.integration.corba.CORBAClientModule`. Note the Java file `AccountProxy.java`.

   Open the subfolder `stubs`. This directory contains CORBA stubs as generated by the `idlj` utility.

Figure 1-94 Code generated from CORBAClientModule

4. Choose **Project>Compile Project** to compile all code.
5. To view the compiled code, use the Explorer [Code Model]. Expand the `ejb` node. The tree looks like this:

Figure 1-95 Generated code from EJB model



Step 6 ? Build the CORBA server

So that you can test that the CORBA connector works, source files are provided for the `account` CORBA server. You can build and run this component locally and call it from a generated Web interface. The rest of this tutorial describes how you can do this.

1. Open a command prompt window (do not close OptimalJ) and navigate to `UserDirectory\sampleprojects\integration\corba\simpleproject`.
2. From the command line, run the following program:

```
idlj -fall account.idl
```

   This generates all skeletons for the CORBA server.

Figure 1-96 Files generated for the CORBA Server

Note: The `idlj.exe` file is located in the `\bin` directory of your JDK installation directory.

3. Return to OptimalJ. Choose **File>Mount Filesystem**. In the wizard, select **Local Directory**, select `UserDirectory\sampleprojects\integration\corba\simpleconnector`, and click **Finish**.
4. In Explorer [Code Model], navigate to `sampleprojects.integration.corba.simpleconnector`.
5. Right-click the file `AccountServer.java` and choose **Compile**.

Step 7 ? Start the CORBA server

1. In the command prompt window, run the `tnameserv.exe` program by entering:

```
tnameserv
```

This starts the CosNaming name service.

Note: The `tnameserv.exe` file is located in the bin directory of your JDK installation directory.

2. Return to OptimalJ. In the Explorer [Code model] and navigate to `sampleprojects.integration.corba.simpleconnector`.
3. Right-click `AccountServer.java` and choose **Execute**.

Step 8 ? Call the CORBA component

Using the generated Web interface for the session bean, invokes the account CORBA connector.

1. Choose **Test>Start Application Server**. Ensure that `..\application\ejb\ejb.jar` is selected in the Archive File Selector window and press **OK**. Wait for the server to start. This starts the Web server and browser, displaying the Main Menu.
2. Click the link **Maintenance Account**. The displayed page shows the operations available on the account CORBA component. There is no stored data, but if you execute the functions in the following order, you can see that the CORBA component is being called. To execute an operation. click on the appropriate link in the menu bar:

- `setInfo`
- `getInfo`
- `deposit`
- `getBalance`
- `withdraw`
- `getBalance`

Alternatively, instead of using a Web interface, you can use an RMI client component. To use this:

1. In the Explorer [Code Model], go to `\sampleprojects\integration\corba\simpleconnector`.
2. Right-click `AccountClient.java` and choose **Compile**.
3. Right-click `AccountClient.java` again and choose **Execute**.
4. The output window shows the AccountServer - IO and the AccountClient - IO.

In this tutorial, you defined and used a CORBA integration model in OptimalJ.

Further reading

For more information, see the documentation on *Integrating with CORBA*.

1.17 Integrating with CICS COBOL via JCA

This tutorial shows how to integrate a CICS COBOL program into OptimalJ using the Java Connector Architecture (JCA). The following files are provided for this tutorial:

- A simple COBOL program called SHIPDATJ. This program takes a date as input, adds 7 days to it, and returns the result.
- A sample main class to show how the COBOL program can be invoked from any Java class.
- A sample RMI client class to show how to invoke the generated session bean.

Prerequisites

Before starting this tutorial, you need to:

- Have an installed CICS region of a version supported by your CICS Transaction Gateway installation
- Ensure that the IBM CICS Transaction Gateway Version 4 or above is installed. It is required for connection to CICS.
- You also need the file `cicseci.rar`, which is not available in the OptimalJ installation kit. The file can be found in the `deployable` directory of the CICS Transaction Gateway installation.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial, you learn how to:.

- Install a JCA resource adapter in OptimalJ
- Define a connection factory for the resource adapter

- Import a CICS COBOL program into OptimalJ
- Generate a JCA component class and session bean for it, and use the generated JCA component to invoke the CICS program

Step 1 ? Prepare the mainframe COBOL program

If the SHIPDATJ program is already available in your CICS system you can skip this step. See your CICS systems programmer for assistance with these tasks.

1. Upload the `shipdatj.cbl` program to a source data set on the mainframe.
2. Compile the program and place the resulting load module in a data set referenced by the DFHRPL DD statement of the CICS region.
3. Use CEDA to define the SHIPDATJ program and install the definition.

Step 2 ? Prepare the file system

During the course of this tutorial, you define models and generate EJB and Web code. This information should be stored in separate directories and mounted for your project.

1. Create a new directory, for example: `\OptimalJ\cobolIntegration`.
2. In this directory, create three subdirectories:

- `\OptimalJ\cobolIntegration\shipcicsModels`
- `\OptimalJ\cobolIntegration\shipcicsEjbCode`
- `\OptimalJ\cobolIntegration\shipcicsWebCode`

Step 3 ? Create new OptimalJ project

Create a new OptimalJ project called `shipcics`.

1. Choose **Project>New OptimalJ Project**. Set the project name to `shipcics` and click **Next**
2. Select **New Model** as the project type and specify the directory `\OptimalJ\cobolIntegration\shipcicsModels` in the **Model dir** field.
3. Enter `shipcics` as the **Fully-qualified package name** and choose `Three Tier Application Structure with Integration` as the **Initial Structure**. Click **Next**.
4. Choose **Mount each filesystem yourself** and click **Next**.

5. Click **Finish**. This mounts the `\OptimalJ\cobolIntegration\shipcicsModels` directory and creates packages for the domain, application, and integration models there.
6. Mount the directories to contain the generated EJB and Web code (`\OptimalJ\cobolIntegration\shipcicsEjbCode` and `\OptimalJ\cobolIntegration\shipcicsWebCode`).

   Choose **File>Mount Filesystem**. In the wizard, select **Local Directory**, and select the directory to mount. Click **Finish**.

Step 4 ? Install the resource adaptor

You need to install the CICS ECI resource adapter in the OptimalJ environment.

1. Choose **Deploy> Install JCA Resource Adapter**.
2. In the **Resource Adapter Filename** field, enter the name of the RAR file containing the resource adapter (or browse to locate the RAR file). The ECI resource adapter is located in the file `cicseci.rar` in the `deployable` directory of your IBM CICS Transaction Gateway installation.
3. Click **Finish** to install the resource adapter. This copies the `cicseci.rar` file to the `resource` directory in your user directory. It also unpacks the file and mounts any jar files within as file systems.

Figure 1-97 Mounted directories and jar files

Step 5 ? Define a CICS ECI connection factory

The JCA component model element created by importing a COBOL program references a connection factory definition. This allows the definition to be used in the generated code and in configuring the EJB server.

1. Choose **Tools>Options** to display the Options panel. Expand the nodes **OptimalJ Configuration>Code Generation>JCA Settings**.
2. Right-click the **Connection Factories** node and choose **Add ECIResourceAdapter**.
3. Enter `cicsecirar` as the **Connection Factory name**. This is the name of the definition and does not have to match the APPLID of the CICS region. Click **OK**.
4. Select the connection factory you just created. If not open already, display the properties. The default properties are automatically supplied.
5. Fill in the relevant properties as follows, and close the Options panel:

Table 1-9 Connection factory properties

Property	Description
ConnectionURL	Address of the CICS Transaction Gateway, for example, `tcp://mycompany.com`
PortNumber	Port used by the Gateway (IBM default is 2006)
ServerName	Server name for the CICS region (APPLID). This field is case-sensitive.

Figure 1-98 Connection factory properties for CICS ECI resource adaptor

For more information on these settings, see *Define a connection factory*.

Step 6 ? Import the COBOL program

1. Choose **Model>Import Model>Import COBOL from File** to start the COBOL import wizard.
2. Select `integration.jca` and click **Next**.
3. Select **Import a CICS Program** and click **Next**.
   Select `shipdatj.cbl` program. This file is located in a subdirectory of your user directory: `UserDirectory\sampleprojects\integration\jca\cics\shipdate\`. The default location of the user directory on Windows is `C:\Documents and Settings\User\.OptimalJ-Edition\3.1`; on UNIX, it is `$HOME`.
4. In the **JCA Connection Factory** field, select the ECI resource adaptor factory you defined earlier.
5. Click **Finish**. Your tree looks like this:

Figure 1-99 Integration model for CICS COBOL

For a description of the integration model elements and how they are mapped from CICS COBOL, see *Structure of JCA integration model (COBOL)*.

Step 7 ? Generate domain and EJB models from integration model

By generating a domain model based on the integration model, you can generate an EJB model including a session bean (and related code) to serve as a wrapper for the JCA component client code and a Web front-end to invoke the session bean.

To generate the domain model and application models from the integration model:

1. Choose **Model>Generate Model>Generate Domain Models>Generate Domain from JCA** to start the wizard.
2. In the wizard, select `shipcics.integration.jca` as the source and `shipcics.domain` as the target domain model. Accept the default values on the last pane and click **Finish**.
3. Choose **Model>Update All Models** to generate the EJB and Web models from the domain model.
4. In the wizard select `shipcics` as the top model package and click **Finish**.

   The EJB and Web models now look like this:

Figure 1-100 EJB and Web application models

Step 8 ? Generate and compile the JCA component code

1. Right-click the `shipcics` package and choose **Generate Code**.
2. When prompted for the location of generated code modules, choose **Mount each filesystem yourself** and click **OK**.

3. When prompted for the EJB module code location, choose `\OptimalJ\cobolIntegration\shipcicsEjbCode`. When prompted for the Web module code location, choose `\OptimalJ\cobolIntegration\shipcicsWebCode`.
4. Choose **Project>Compile Project**to compile the code.

Figure 1-101 Code model

The jca package now contains a `ShipdatjJCAComponent.java` source file and a `recordshipdatj` package directory. The recordshipdatj package contains a `Dfhcommarea.java` class. The ShipdatjJCAComponent class contains code to invoke the JCA interactions defined in the model. The JCA component class can be used as a normal class to invoke the SHIPDATJ CICS COBOL program.

In the ejb package, you have a `ShipdatjBean.java` and other files, defining a session bean. The session bean uses the JCA component class in the jca package to invoke the shipdatj CICS COBOL program.

In the web package, you have forms, actions and other files, defining a web front end. The web front end uses the session bean in the ejb package to invoke the SHIPDATJ CICS COBOL program.

Step 9 ? Create the JCA component test program

The JCA component test program uses the generated JCA component in the jca package to invoke the SHIPDATJ COBOL program. Steps 10 and 11 are optional, but provide an easy environment in which to debug the generated component. This can be helpful in diagnosing problems when the CICS program is not being executed.

1. Mount the directory `UserDir\sampleprojects\`.
2. In the Explorer [Code Model], copy `integration\jca\cics\shipdate\ShipdateNonManagedJCATest.java` from this directory to the `shipcics` package in directory `\OptimalJ\cobolIntegration\shipcicsModels`.
3. If your CICS region uses a code page other than the default Cp1140 (compatible with EBCDIC 037), open the file in the OptimalJ editor and change the encoding defined by the variable *characterEncoding*. (See the character encoding sample available in the directory `characterencoding` for additional information.)
4. Save your changes and compile the `ShipdateNonManagedJCATest` source.

Step 10 ? Test the JCA component

1. In the Explorer [Code Model], navigate to the folder `\OptimalJ\cobolIntegration\shipcicsModels\shipcics`.
2. Right-click the file `ShipdateNonManagedJCATest.java` and choose **Execute**.
3. The Output Window displays:

```
Welcome to the Optimal Delivery Shipping System!
Your order date is Oct 5, 2001
Please wait while we calculate your shipping date...
Your order will ship on Oct 12, 2001
```

Step 11 ? Test the session bean

You can use the generated Web application to invoke the JCA component via the session bean generated from the EJB model.

1. Choose **Test>Start Application Server**. Ensure that `..\application\ejb\ejb.jar` is selected in the Archive File Selector window and press **OK**. Wait for the server to start. This starts the Web server and browser, displaying the Main Menu.
2. Click **Maintenance Shipdatj**.
3. Click **shipdatj**.
4. Enter the day, month, and year of the order date and click OK.

   The page redisplays with a ship date that is 7 days after the entered order date.

Instead of using the Web application, you can use the supplied RMI test program. This program uses the session bean, generated from the EJBSessionComponent, to invoke the generated JCA component, which in turn invokes the SHIPDATJ COBOL program.

Create the test program and test it:

1. In the Explorer [Code Model], copy the file `ShipdateBeanTest.java` from`\sampleprojects\integration\jca\cics\shipdate` to the `\OptimalJ\cobolIntegration\shipcicsModels` directory.
2. Right-click `ShipdateBeanTest.java` and choose **Compile**.
3. Choose **Test>Start Application Server**. Ensure that `..\application\ejb\ejb.jar` is selected in the Archive File Selector window and press **OK**. Wait for the server to start. This starts the Web server and browser, displaying the Main Menu.
4. Right-click `ShipdateBeanTest` and choose **Execute**.
5. The Output Window should display

   ```
   Welcome to the Optimal Delivery Shipping System!
   Your order date is Oct 5, 2001
   Please wait while we calculate your shipping date...
   Your order will ship on Oct 12, 2001
   ```

   Common errors are:

   ```
   java.io.IOException: ShipstatJCAConnection.shipstat : Exception
     executing interaction:
   javax.resource.spi.CommException: CTG9630E: IOException
     occurred in communication with CICS.:
   linked exception : java.io.IOException: CCL6651E: Unable
     to connect to the Gateway.
   [address = ibm1.mycompany.com, port = 2006]
   [java.net.UnknownHostException:
   ```

```
ibm1.mycompany.com]
```

The JCA component was unable to connect to the specified host (ibm1.mycompany.com). Check the ConnectionURL setting in the Options panel.

```
java.io.IOException: ShipstatJCAConnection.shipstat : Exception
  executing interaction:
javax.resource.spi.CommException: CTG9630E: IOException
  occurred in communication with CICS.:
linked exception : java.io.IOException: CCL6651E: Unable
  to connect to the Gateway.
[address = cw01.compuware.com, port = 2006] [java.net.ConnectException:
  Connection refused: connect]
```

The JCA component was unable to connect to the CICS Transaction Gateway on the specified port (2006). Make sure the gateway has been started and that the ConnectionURL and PortNumber specified in the Options panel are correct.

```
java.io.IOException: ShipstatJCAConnection.shipstat : Exception
  executing interaction:
javax.resource.spi.CommException: CTG9631E: Error occurred
  during interaction with CICS.
Error Code=ECI_ERR_NO_CICS
```

The gateway was unable to connect to the CICS region. Make sure the name specified in the *ServerName* field in the Connection Factory settings panel matches a server name defined to the CICS Transaction gateway.

Messages beginning with CCL or CTG and return codes beginning with ECI_ originate in the CICS Transaction Gateway. See the gateway documentation for further information.

In this tutorial you have integrated a sample CICS COBOL program with an OptimalJ application.

A second sample, shipstat, is available in the directory `UserDir\sampleprojects\integration\jca\cics\shipstat\`. The steps for this example are available in the OptimalJ scenarios. Choose **Scenarios>Integrating with CICS**.

Further reading

For more information, see the documentation on *Integrating with JCA* and the CICS Transaction Gateway documentation.

1.18 Integrating with CICS COBOL via JMS

This tutorial shows how to integrate a CICS COBOL program into OptimalJ using the Java Messaging Service (JMS) and WebSphere MQ.

Note: The purpose of this tutorial is to help you understand the message bridge support. Although messaging is typically used for a loosely-coupled solution, the example shows a tightly-coupled use of messaging. You should consider using JCA for tightly-coupled interaction with CICS.

The following files are provided for this tutorial:

- A simple COBOL program called SHIPDATJ. This program takes a date as input, adds 7 days to it, and returns the result.
- A sample client class to show how to invoke the generated session bean and receive the message sent by the generated message driven bean.

Prerequisites

Before starting this tutorial, you need to:

- Ensure an installed CICS region is configured to use the WebSphere MQ CICS Bridge.
- Determine the name of the bridge queue defined to the region, as well as the queue manager name, host name and port number.
- Determine the name a queue where the bridge will write the reply message.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial, you learn how to:

- Install WebSphereMQ Classes for JMS into OptimalJ.
- Define a connection factory and a queue for WebSphere MQ.
- Import a CICS COBOL program.
- Define a MessageBridgeMessage.

- Define EJB an EJB session component and JMS message components.
- Generate code from the EJB and message bridge models.
- Use the generated code to invoke the CICS program.

Step 1 ? Prepare the mainframe COBOL program

If the SHIPDATJ program is already available in your CICS system you can skip this step. See your CICS systems programmer for assistance with these tasks.

1. Upload the `shipdatj.cbl` program to a source data set on the mainframe. This program can be found in `YourUserDir\sampleprojects\integration\jca\cics\shipdate`
2. Compile the program and place the resulting load module in a data set referenced by the DFHRPL DD statement of the CICS region.
3. Use CEDA to define the SHIPDATJ program and install the definition.

Step 2 ? Prepare the file system

During the course of this tutorial, you define models and generate EJB and Web code. This information should be stored in separate directories and mounted for your project.

1. Create a new directory, for example: `\OptimalJ\messagebridge`.
2. In this directory, create two subdirectories:

- `\cicsbridgeModel`
- `\cicsbridgeEjbCode`

Step 3 ? Create new OptimalJ project

Create a new OptimalJ project called `cicsbridge`.

1. Choose **Project>New OptimalJ Project**. Set the project name to `cicsbridge` and click **Next**
2. Select **New Model** as the project type and specify the directory `\OptimalJ\messageBridge\cicsbridgeModel` in the **Model dir** field.
3. Enter `cicsbridge` as the **Fully-qualified package name** and choose `Three Tier Application Structure with Integration` as the **Initial Structure**. Click **Next**.

4. Choose **Mount each filesystem yourself** and click **Next**.
5. Click **Finish**. This mounts the `\OptimalJ\messageBridge\cicsbridgeModel` directory and creates packages for the domain, application, and integration models there.
6. Mount the `\OptimalJ\messageBridge\cicsbridgeEjbCode` directory to contain the generated EJB code.

   Choose **File>Mount Filesystem**. In the wizard, select **Local Directory**, and select the directory to mount. Click **Finish**.

Step 4 ? Install the WSMQ Classes for JMS in OptimalJ

The WSMQ Classes for JMS are included as part of WebSphereMQ. They can also be downloaded as part of *SupportPac MA88*. The JAR files are in the `lib` directory of your WebSphereMQ installation.

1. Choose **Deploy>Install WSMQ Jar Files**
2. In the **WSMQ Jar Directory** field, enter the path to the `\lib` directory in your WSMQ installation or other location where the JAR files are located.
3. Click **Finish** to install the jar files.

Step 5 ? Define a WSMQ Queue connection factory

Create a WSMQ Queue connection factory definition. This is used in the generated code and in configuring the EJB server.

Later in this tutorial, you will create an EJB session component and message-driven component model elements that reference this definition.

1. Choose **Tools>Options** to display the Options panel. Expand the nodes **OptimalJ Configuration>Testing>JMS**.
2. Right-click the **WSMQ Queue Connection Factories** node and choose **Add WSMQ Queue Connection Factory**.
3. Enter `wsmqqcf` as the name for the connection factory. This name is required to use the WebSphere MQ configuration in the internal JBoss in OptimalJ. Click **OK**.
4. Select the connection factory you just created. If not open already, display the properties. The default properties are automatically supplied.
5. Fill in the relevant properties as follows, and close the Options panel:

Table 1-10 WSMQ Queue Connection Factory properties

Property	Description
ChannelName	The channel name used to communicate with the server, for example, `SYSTEM.DEF.SVRCONN`. This field is case-sensitive.
HostName	Address of the queue manager, for example, `mycompany.com`.
Password	Can be used to provide a password to WSMQ if required.
Port	Port used by the queue manager (IBM default is 1414).
QueueManagerName	Name of the queue manager, for example `M530`. This field is case-sensitive.
User	Can be used to provide a user ID to WSMQ if required.

Figure 1-102 Properties for WSMQ Queue Connection Factory

Step 6 ? Define WSMQ Queues

Define two WSMQ queues?one defined to the CICS region, which will receive a CICS Bridge message, and one where the CICS Bridge will send the response.

The EJB session component (which you create in step 14) sends a CICS Bridge message to a WSMQ queue defined to the CICS region. This message contains the name of a reply-to queue where the CICS Bridge sends the response.

The EJB message-driven component (which you create in step 12) processes messages from the reply-to queue.

1. Right-click the **WSMQ Queues** node in the Options panel and choose **Add WSMQ Queue** from the pop-up menu.
2. Enter the name of the bridge queue as defined to the CICS region, for example `CICS.BRIDGE.QUEUE`. Note that queue names are case sensitive. Click **OK**.
3. Select the queue you just created. If not open already, display the properties. The default properties are automatically supplied.
4. Fill in the relevant properties as follows, and close the Options panel:

Table 1-11 WSMQ Queue properties

Property	Description
CCSID	Character set to be used to encode text strings in messages sent to this destination.
Encoding	Specifies the encoding to be used for numeric fields in messages sent to this destination.
QueueManagerName	Name of the queue manager where this queue is defined, for example `MMQM`. This field is case-sensitive.
UseRFH2	Defines whether messages sent to the queue should contain a version 2 rules and formatting header. Set to `false` for messages intended for the WSMQ CICS Bridge.

Figure 1-103 Properties for WSMQ Queue Connection Factory

5. Repeat the steps above to define the reply-to queue. The reply-to queue must be a different queue (for example,

CICS.BRIDGE.REPLY.QUEUE) from the bridge queue, but must be defined in the same queue manager.

Step 7 ? Define a JMS Queue connection factory

Define a connection factory for the default JMS provider. The connection factory is used to send a message back to the sample client.

A message-driven component model element (which you create in Step 12) references this connection factory definition. The definition is used in the generated code and in configuring the EJB server.

1. Right-click the **Queue Connection Factories** node in the Options panel and choose **Add Queue Connection Factory**.
2. Enter `clientqcf` as the name for the connection factory. This name is required because the sample client code refers to this name. Click **OK**.

Step 8? Define a JMS Queue

In this step, you define a queue for the default JMS provider. A message-driven bean sends a message to this queue when it receives the reply-to message from the CICS bridge. The sample client reads messages from this queue.

A message-driven component model element (which you will create in Step 12) references this queue. The queue is used in the generated code and in configuring the EJB server.

1. Right-click the Queue node in the Options panel and choose **Add Queue**.
2. Enter `clientq` as the name for the connection factory. This name is required because the sample client code refers to this name. Click **OK**.
3. Click **Close**.

Step 9 ? Import the COBOL program

Importing a COBOL program into the message bridge integration model creates a COBOLSchema in the message bridge model.

1. Choose **Model>Import Model>Import COBOL from File** to start the COBOL import wizard.
2. Select `integration.messagebridge` and click **Next**.
3. Select `shipdatj.cbl` program. This file is located in a subdirectory of your user directory: `UserDirectory\sampleprojects\integration\jca\cics\`

`shipdate\`. The default location of the user directory on Windows is `C:\Documents and Settings\User\.OptimalJ-Edition\3.1`; on UNIX, it is `$HOME`.

4. Click **Finish**. Your tree looks like this:

Figure 1-104 Message Bridge model for CICS COBOL

For a description of the integration model elements and how they are mapped from CICS COBOL, see *Structure of JCA integration model (COBOL)*.

Step 10 ? Create a MessageBridgeMessage

To complete the message bridge integration model, you need to define a MessageBridgeModule and MessageBridgeMessage. The MessageBridgeModule specifies the message bridge components that will be grouped in a deployable unit, and the MessageBridgeMessage is one of those components, representing the payload of a JMSMessage.

1. Right click `integration.messagebridge` in Explorer [Application Model] and choose **New Child>MessageBridgeMessage** to start the wizard.
2. Name the message `ShipdatjMessage` and click **Next**.
3. Click **Create New** to start a wizard to create a new MessageBridgeModule.
4. Name the module `shipdatj` and click **Finish**.
5. Select the `shipdatj` MessageBridgeModule and click **Next**.
6. Enter `SHIPDATJ` as the Program Name and click **Next**.

Note: This field is case-sensitive. If in doubt, use all upper case letters.

7. Select `integration.messagebridge.recordshipdatj.DFHCOMMAREA` and click **Finish**. Your tree looks like this:

Figure 1-105 Message Bridge model for CICS COBOL

For a description of the integration model elements see *Structure of the Message Bridge integration model*.

Step 11 ? Generate EJB model from integration model

By generating an EJB model based on the integration model, you can generate a JMS message that uses the MessageBridgeMessage and an EJBStruct that represents the COMMAREA.

To generate the EJB model from the integration model:

1. Choose **Model>Generate Model>Generate Application Models>Generate EJB from MessageBridge** to start the wizard.
2. In the wizard, select `cicsbridge.integration.messagebridge` as the source and `cicsbridge.application.ejb` as the target model. Accept the default values on the last pane and click **Finish**.

   Your tree now looks like this:

Figure 1-106 EJB model for CICS COBOL

Step 12 ? Set the replyTo properties on a JMSMessage

You need to couple the JMSMessage definition with the WMSQ Queue definition.

1. In Explorer [Application Model], select the JMSMessage `application.ejb.ShipdatjMessage`.
2. Set the *replyToDestination* in the properties panel to the name of the reply-to WSMQ queue defined in Step 6. For example, `CICS.BRIDGE.REPLY.QUEUE`.
3. Set the *replyToDestinationType* property to `Queue`.

Step 13 ? Create JMSMessages for the client

1. Right click `application.ejb` in Explorer [Application Model] and choose **New Child>JMSMessage** to start the wizard.
2. Name the message `ShipdatjReply` and click **Next**.
3. Select `Object Message` as the **JMS Message Type** and `EJBStructType` as the **Transported Object**. Click **Next**.
4. Select `application.ejb.recordshipdatj.Dfhcommarea` and click **Finish**.
5. Repeat the steps above to create a JMSMessage named `ShipdatjError`, this time selecting `Text Message` as the **JMS Message Type**.

Step 14 ? Create an EJBSessionComponent

To initiate messages to the CICS COBOL program via the message bridge, you need to define an EJBSessionComponent.

1. Right click `application.ejb` in Explorer [Application Model] and choose **New Child>EJBSessionComponent** to start the wizard.
2. Name the component `ShipdatjSB` and click **Next**.
3. Select **Create Without a Serving Attribute** and click **Next**.
4. Click **Next** to accept the available interfaces and state.
5. Select the `shipdatj` EJBModule and click **Next**.
6. Click the top **Add** button to create a new Business Method. Name the method `sendMessage`.
7. Click the bottom **Add** button to create a new parameter. Name the parameter `in` and select `application.ejb.recordshipdatj.Dfhcommarea` as the **Type**.
8. Click **Next** to skip the **Used Components** panel.
9. Click **Add** to add a produced message. Select `application.ejb.ShipdatjMessage` and click **OK**. Click **Next**.
10. Click **Next** to skip the **Topics to Produce To** panel.
11. Click the top **Add** button to add a Queue. Name the queue the same as the CICS bridge queue defined in Step 6. For example, `CICS.BRIDGE.QUEUE`.

    Click the bottom button to add a Queue Connection Factory. Name the connection factory `wsmqqcf` as defined in Step 5. Click **Finish**.

Step 15 ? Create an EJBMessageDrivenComponent

1. Right click `application.ejb` in Explorer [Application Model] and choose **New Child>EJBMessageDrivenComponent** to start the wizard.
2. Name the component `ShipdatjMB` and click **Next**.
3. Enter the name of the reply-to queue defined in Step 6 and select `Queue` as the **Destination Type**. Click **Next**.
4. Set the type of delivery to `Durable` and click **Next**.
5. Select the `shipdatj` EJBModule and click **Next**.
6. Select `application.ejb.ShipdatjMessage` as the message to consume. Click **Next** three times to skip the Message Selector and Used Components panels.

7. On the Produced Messages panel, click the **Add** button, select the `application.ejb.ShipdatjReply` message and click **OK**. Repeat this step, selecting the `application.ejb.ShipdatjError` message. Click **Next** twice to skip the Topics panel.
8. Use the **Add** buttons to add a queue named `clientq` and a queue connection factory named `clientqcf`. Click **Finish**. Your tree looks like this:

Figure 1-107 EJB application model

Step 16 ? Generate the application code

1. Right-click the `cicsbridge` package Explorer [Application Model] and choose **Generate Code**.

   When prompted for the shipdatj module code location, choose `\OptimalJ\messageBridge\cicsbridgeEjbCode`.

Figure 1-108 Code model

The messagebridge package now contains a package directory. The recordshipdatj package contains a `Dfhcommarea.java` class that represents the COBOL COMMAREA.

In the ejb package, you have a `ShipdatjSBBean.java` and other files defining a session bean. The session bean contains a helper method to send the ShipdatjMessage, which will cause WSMQ to invoke the SHIPDATJ CICS COBOL program.

You also have a `ShipdatjMB.java`, defining a message-driven bean. The message-driven bean contains an `onMessage` method to process the reply-to message. It also contains helper methods to send the ShipdatjReply and ShipdatjError messages.

Step 17 ? Add code to free blocks and compile

The generated session bean and message-driven bean contain helper methods to send the messages defined in the model. The session bean contains an empty `sendMessage` business method. The message-driven bean contains an `onMessage` method to process the received reply-to message. You need to add code in free blocks to connect the message production to the other methods.

1. In Explorer [Application Model] right-click the ShipdatjSB session component and select **Edit Free Blocks in Generated Files>Business Methods>ShipdatjSBBean.java>body (sendMessage)**.
2. Insert the following code at the cursor in the Source window:

```
try {
    produceShipdatjMessage("CICS.BRIDGE.QUEUE",
       Queue.class, "wsmqqcf", in);
} catch (Exception e) {
    logger.error(e.toString());
}
```

Note: Replace the string `CICS.BRIDGE.QUEUE` with the name of the bridge queue defined in Step 6.

3. Press Ctrl+S to save your changes.
4. Right-click the ShipdatjMB message driven component and select **Open ShipdatjMB.java**
5. Locate the `consumeShipdatjMessage` method in the Source window and find the comment `Processes a successful reply`. Insert the following code after the comment:

```
logger.error("Sending ShipdatjReply message");
produceShipdatjReply("clientq", Queue.class, "clientqcf",
   dfhcommarea);
```

6. Find the comment `Processes an unsuccessful reply` and insert the following code after it:

```
logger.error(errorMessage);
produceShipdatjError("clientq", Queue.class, "clientqcf",
   errorMessage);
```

7. Press Ctrl+S to save your changes.
8. Choose **Project>Build Project**.

Step 18 ? Create the message bridge client test program

The message bridge client test program invokes the `sendMessage` method in the generated session bean to send a CICS Bridge message requesting that the SHIPDATJ COBOL program be run. It then tries to receive a message from the `clientq` queue that either contains the output COMMAREA or an error message.

1. Copy the file `ShipdateMessageTest.java` to the `messageBridge\cicsbridgeEjbCode\cicsbridge` directory and open it in the OptimalJ editor. This file is located in the directory `UserDirectory\sampleprojects\integration\messagebridge\cics\shipdate`. Right-click the `cicsbridge` directory and select **Refresh Folder**.
2. Right-click `ShipdateMessageTest.java` in Explorer [Code Model] and choose **Compile**.
3. Choose **Test>Start Application Server**. Ensure that `application\ejb\shipdatj.jar` is selected in the Archive File Selector window and click **OK**. Wait for the server to start.
4. Right-click `ShipdateMessageTest` and choose **Execute**.
5. The output window should display

```
Welcome to the Optimal Delivery Shipping System!
Your order date is Oct 5, 2001
Please wait while we calculate your shipping date...
Your order will ship on Oct 12, 2001
```

In this tutorial you have integrated a sample CICS COBOL program with an OptimalJ application using JMS and WebSphere MQ.

Further reading

For more information, see *Integrating with Message Bridges* and *Integrating with JCA*, as well as the WebSphere MQ documentation.

1.19 Integrating with IMS COBOL

This tutorial shows you how to integrate an IMS COBOL program into OptimalJ. The following files are provided for this tutorial:

- A simple COBOL program called SHIPDATJ. This program takes a date as input, adds 7 days to it, and returns the result.
- COBOL copybooks defining the input and output messages for the IMS program.
- A sample main class to show how the COBOL program can be invoked from any Java class.
- A sample RMI client class to show how to invoke the generated session bean.

Prerequisites

Before starting this tutorial, ensure that the following are installed:

- IBM IMS Connector for Java 1.2.2 or above can be installed on a variety of platforms. The IMS Connector for Java installation includes the IMS resource adapter file `imsico.rar`, which must be installed in OptimalJ as described in the tutorial. This file is not part of the OptimalJ installation kit.
- IMS Transaction Manager and IMS Connect must be installed on the mainframe.

These are required for the connection to IMS.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial, you learn how to:

- Install a JCA resource adapter in OptimalJ.
- Define a connection factory for the resource adapter.
- Create the model elements to represent the IMS transaction.
- Import COBOL copybooks representing IMS messages into OptimalJ.
- Generate a JCA component class and session bean.
- Use the generated code to invoke the IMS transaction.

Step 1 ? Prepare the mainframe COBOL program

If the SHIPDATJ transaction is already available in your IMS system you can skip this step. See your IMS systems programmer for assistance with these tasks.

1. Upload the shipdatj.cbl program and copybooks to a source data set on the mainframe.
2. Compile the program and place the resulting load module in a data set referenced by the STEPLIB DD statement of an IMS message processing region.
3. Define and generate a PSB for the program with the name SHIPDATJ.
4. Perform an ACBGEN using the generated PSB.
5. Define the application named SHIPDATJ using the PSB, and a transaction code SHIPDATJ.
6. Generate the IMS system.

Step 2 ? Prepare the file system

During the course of this tutorial, you define models and generate EJB and Web code. This information should be stored in separate directories and mounted for your project.

1. Create a new directory, for example: `\OptimalJ\imsIntegration`.
2. In this directory, create three subdirectories:

- `\OptimalJ\imsIntegration\shipimsModels`
- `\OptimalJ\imsIntegration\shipimsEjbCode`
- `\OptimalJ\imsIntegration\shipimsWebCode`

Step 3 ? Create a new OptimalJ project

Create a new OptimalJ project called `shipping`.

1. Choose **Project>New OptimalJ Project**. Set the project name to `shipping` and click **Next**
2. Select **New Model** as the project type and specify the directory `\OptimalJ\imsIntegration\shipimsModels` in the **Model dir** field.
3. Enter `shipims` as the **Fully-qualified package name** and choose `Three Tier Application Structure with Integration` as the **Initial Structure**. Click **Next**.
4. Choose **Mount each filesystem yourself** and click **Next**.
5. Click **Finish**. This mounts the `\OptimalJ\imsIntegration\shipimsModels` directory and creates packages for the domain, application, and integration models there.

6. Mount the directories to contain the generated EJB and Web code (`\OptimalJ\imsIntegration\shipimsEjbCode` and `\OptimalJ\imsIntegration\shipimsWebCode`).
   Choose **File>Mount Filesystem**. In the wizard, select **Local Directory**, and select the directory to mount. Click **Finish**.

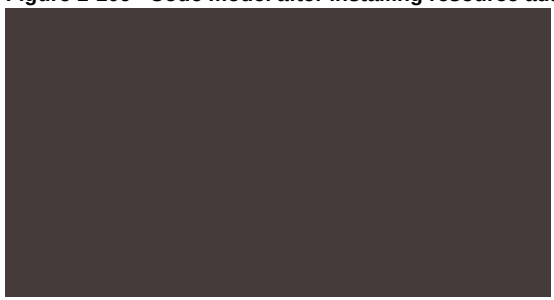
Step 4 ? Install the resource adaptor

You need to install the IMS Connector for Java resource adapter in the OptimalJ environment.

To set up your environment:

1. Choose **Deploy>Install JCA Resource Adapter**.
2. In the **Resource Adapter Filename** field, enter or browse to the RAR file containing the resource adapter. The IMS resource adapter is located in the `imsico.rar` file in your IBM IMS Connector for Java installation.
3. Click **Finish** to install the resource adapter. This copies the `imsico.rar` file to the `resource` directory in your user directory. It also unpacks the RAR file and mounts any JAR files within as file systems.

Figure 1-109 Code model after installing resource adaptor



Step 5 ? Define an IMS Connector for Java connection factory

The JCA component model element that you will create later references a connection factory definition. This allows the definition to be used in the generated code and in configuring the EJB server.

1. Choose **Tools>Options** to display the Options window. Browse to **OptimalJ>Configuration>Code Generation>JCA Settings**.
2. Right-click **Connection Factories** and choose **Add IMS Connector for Java**.

3. Enter `ims7` as the name for the connection factory. This is the name of the definition and does not have to match the ID of the IMS system. Click **OK**.
4. Select the connection factory you just created. If not open already, display the properties. The default properties are automatically supplied.
5. Fill in the relevant properties for your IMS Connect, as follows:

Table 1-12 Connection factory properties

Property	Description
HostName	Address of host where IMS Connect runs, for example, `mycompany.com`
PortNumber	Port used by IMS Connect
DataStoreName	The name of the target IMS datastore. This must match the ID parameter of the Datastore statement that is specified in the IMS Connect configuration member when IMS Connect is installed. This field is case-sensitive.

Figure 1-110 IMS connection factory properties

For more information on these settings, see *Define a connection factory*.

Step 6 ? Create the JCA client module and JCA component

1. In Explorer [Application Model], expand the nodes `shipims.integration.jca`.
2. Right-click the jca package and choose **New Child>JCAClientModule**. Click Finish to accept the default name.
3. Right-click the jca package again and select **New Child>JCAComponent**.
4. Select the JCAClientModule and click **Next**.
5. Enter `Shipdatj` as the name of the JCAComponent. Select the connection factory you defined earlier and click **Next**.
6. The name of the JCAInteraction defaults to `shipdatj`. This name (after conversion to upper case) must match the name of the transaction as defined to IMS.
7. Click **Add** twice to add one in parameter and one return parameter. You can ignore the parameter types for now.
8. Click **Finish**.

Step 7 ? Import the COBOL message structures

1. Choose **Model>Import Model>Import COBOL from File** to start the COBOL import wizard.
2. Select `integration.jca` and click **Next**.
3. Select **Import a Copybook** and click **Next**.
4. Use the browse button to select `shipin.cpy` copybook. This file is located in `UserDir \sampleprojects\integration\jca\ims\shipdate\`. Click **Next**.
5. Select the `in` parameter under the `shipdatj` interaction and click **Finish**.
6. Perform the same steps to import the `shipout.cpy` copybook, this time selecting the `ret` parameter.
7. Your tree looks like this:

Figure 1-111 Integration model after importing COBOL copybooks

Step 8 ? Generate domain and application models from integration model

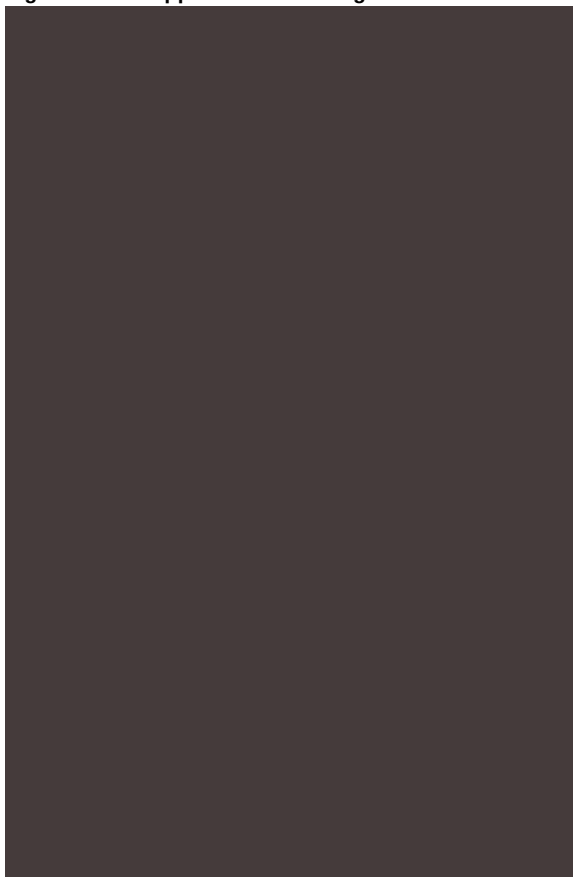
By generating an application model based on the integration model, you can generate a session bean (and related code) to serve as a wrapper for the JCA component client code. By first generating a domain model based on the integration model, you can also generate a Web front end to invoke the session bean.

To generate the domain model and application models from the integration model:

1. Choose **Model>Generate Model>Generate Domain Models>Generate Domain from JCA** to start the wizard.
2. In the wizard, select `shipims.integration.jca` as the source and `shipims.domain` as the target domain model. In the last pane, accept the defaults and click **Finish**.
3. Choose **Model>Update All Models** to generate the EJB and Web models from the domain model.
4. In the wizard, select `shipims` as the top model package and click **Finish**.

   Your tree now looks like this:

Figure 1-112 Application and integration models



Step 9 ? Generate and compile the code

1. Right-click the `shipims` package and choose **ModelGenerate Code**.
2. When prompted for the location of generated code modules, choose `\OptimalJ\imsIntegration\shipimsEjbCode` for the EJB module code and `\OptimalJ\imsIntegration\shipimsWebCode` for the Web module code.
3. Choose **Project>Compile Project** to compile all the code.

Figure 1-113 Code model after compiling code

The jca package now contains a `ShipdatjJCAComponent.java` source file and package directories for `copybookshipin` and `copybookshipout`. The package directories contain classes that extend the Record framework. The ShipdatjJCAComponent class represents the JCA component and can be used as a normal class to invoke the SHIPDATJ IMS COBOL program.

In the ejb package, you have a `ShipdatjBean.java` and other files defining a session bean. The session bean uses the JCA component class in the jca package to invoke the SHIPDATJ IMS COBOL program.

In the web package, you have forms, actions and other files, defining a Web front-end. The Web front-end uses the session bean in the ejb package to invoke the SHIPDATJ IMS COBOL program.

Step 10 ? Create the JCA component test program

The JCA component test program uses the generated JCA component class in the jca package to invoke the SHIPDATJ COBOL program. Steps 11 and 12 are optional, but provide an easy environment in which to debug the generated component. This can be helpful in diagnosing problems when the IMS transaction is not being executed.

1. Mount the directory `UserDirectory\sampleprojects\`.
2. In the Explorer [Code Model], copy `integration\jca\ims\shipdate\ShipdateNonManagedJCATest.java` from this directory to the `shipims` package in directory `\OptimalJ\imsIntegration\shipimsModels`.
3. If your IMS system uses a code page other than the default Cp1140 (compatible with EBCDIC 037), open the file and change the encoding defined by the variable `characterEncoding`. (See the character encoding sample available in the directory `characterencoding` for additional information.)

   Save your changes and compile the ShipdatjNonManagedJCATest source.

Step 11 ? Test the JCA component

1. In the Explorer [Code Model], navigate to the folder `\OptimalJ\imsIntegration\shipimsModels`.
2. Right-click the file `ShipdateNonManagedJCATest.java` and choose **Execute**.
3. The output window displays:

```
Welcome to the Optimal Delivery Shipping System!
Your order date is Oct 5, 2001
Please wait while we calculate your shipping date...
Your order will ship on Oct 12, 2001
```

Step 12 ? Test the session bean

You can use the generated Web application to invoke the JCA component via the session bean generated from the EJB model.

1. Choose **Test>Start Application Server**. Ensure that `..\application\ejb\ejb.jar` is selected in the Archive File Selector window and click **OK**. Wait for the server to start. This starts the Web server and browser, displaying the Main Menu.
2. Click **Maintenance Shipdatj**.
3. Click **shipdatj**.

4. Enter the day, month, and year of the order date and click **OK**. The page redisplays with a ship date that is 7 days after the entered order date.

Instead of using the Web application, you can use the supplied RMI test program. This program uses the session bean, generated from the EJBSessionComponent, to invoke the generated JCA component, which in turn invokes the SHIPDATJ COBOL program.

1. Create the test program:
   - Copy the file `ShipdatjBeanTest.java` to your shipims package directory. This file is located in the directory `UserDir\sampleprojects\integration\jca\ims\shipdate`.
   - Right-click `ShipdatjBeanTest.java` and choose **Compile**.
2. In Explorer [Code Model], select the `shipims.application.ejb` package.
3. From the menu bar, choose **Test>Start Application Server**. Ensure that `..\application\ejb\ejb.jar` is selected in the Archive File Selector window and press **OK**. Wait for the server to start.
4. Right-click ShipdatjBeanTest and choose **Execute**.
5. The output window should display

```
Welcome to the Optimal Delivery Shipping System!
Your order date is Oct 5, 2001
Please wait while we calculate your shipping date...
Your order will ship on Oct 12, 2001
```

Common errors are:

```
 1 | java.io.IOException: ShipstatJCAComponent.shipstat :
 2 | Exception executing interaction:
 3 | javax.resource.spi.CommException: ICO0003E:
 4 | com.ibm.connector2.ims.ico.IMSManagedConnection@c5c3ac.
   |   connect(String,
 5 | Integer) error. Failed to connect to host [ibm1.myhost.
   |   com], port [1234].
 6 | [java.net.ConnectException: Connection refused: connect]
```

The resource adapter was unable to connect to IMS Connect on the specified port (1234). Make sure IMS Connect is available and that the HostName and PortNumber specified in the Options panel are correct.

```
1 | java.io.IOException: ShipstatJCAComponent.shipstat :
2 | Exception executing interaction:
3 | javax.resource.spi.CommException: ICO0003E:
4 | com.ibm.connector2.ims.ico.IMSManagedConnection@c5c3ac.
  |   connect(String,
5 | Integer) error. Failed to connect to host [ibm1.mycompany.
  |   com], port [43,215].
6 | [java.net.UnknownHostException: ibm1.mycompany.com]
```

The resource adapter was unable to connect to the specified host (ibm1.mycompany.com). Check the HostName setting in the Options panel.

```
1 | java.io.IOException: ShipstatJCAComponent.shipstat :
2 |  Exception executing interaction:
3 | javax.resource.spi.EISSystemException: ICO0001E:
4 | com.ibm.connector2.ims.ico.IMSManagedConnection@c5c3ac.
  |   call(Connection, InteractionSpec,
5 | Record, Record) error. IMS Connect returned error: RETCODE=[4],
  |    REASONCODE=[NFNDDST ].
6 | [com.ibm.ims.ico.IMSAdapter@fcf0ce.receive(InteractionSpec):
7 |  com.
  |    ibm.ims.ico.IMSConnResourceException]
```

The specified data store name was not defined to IMS Connect. Check the DataStoreName setting in the Options panel.

Messages beginning with ICO originate with the IMS Connector for Java resource adapter. RETCODE and REASONCODE codes originate from IMS Connect. See the corresponding IMS documentation for further information.

```
java.io.IOException: ShipdatjJCAComponent.shipdatk :
Exception executing interaction: com.ibm.connector2.ims.
  ico.IMSDFSMessageException:
ICO0075E: com.ibm.ims.ico.IMSAdapter@cee361.receive(InteractionSpec)
  error.
IMS returned DFS message: DFS064 07:46:24 DESTINATION CAN
  NOT BE FOUND OR CREATED
```

The specified transaction code is not defined. Verify that the transaction is defined to IMS and that the name of the JCAInteraction in the model matches the transaction code. Note that

if the JCAInteraction name is changed, the JCAComponent must be regenerated and all code recompiled.

In this tutorial, you have integrated a sample IMS COBOL program with an OptimalJ application.

A second sample, SHIPSTAT, is available in the directory `UserDir\sampleprojects\integration\jca\ims\shipstat`. The steps for this example are available from the OptimalJ scenarios. Choose **ScenariosIntegrating with IMS**.

Further reading

For more information on integrating with IMS, see the documentation under *Integrating with JCA* and IMS Connect documentation.

1.20 Handling the COBOL REDEFINES clause

This tutorial shows different ways to handle cases where REDEFINES is used in a record imported into OptimalJ.

A simple COBOL program called REDEF is provided for this tutorial. This program takes a function code as input and can return two types of records. The function code can be A or B, and the corresponding record data is returned along with the current date.

Prerequisites

Before starting this tutorial, you need to:

- Have an installed CICS region of a version supported by your CICS Transaction Gateway installation
- Ensure that the IBM CICS Transaction Gateway Version 4 or above is installed. It is required for connection to CICS.
- You also need the file `cicseci.rar`, which is not available in the OptimalJ installation kit. The file can be found in the `deployable` directory of the CICS Transaction Gateway installation.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial, you import a CICS COBOL program into OptimalJ, generate a JCA component class, session bean and web interface for it, and use the generated JCA component to invoke the CICS program.

In this tutorial, you learn how to:

- Install a JCA resource adapter in OptimalJ
- Define a connection factory for the resource adapter
- Import a CICS COBOL program
- Update the JCA model to select the definitions of fields desired in the user interface
- Generate code to invoke the CICS program
- Update the generated JSP to display the desired output record

Step 1 ? Prepare the mainframe COBOL program

If the REDEF program is already available in your CICS system you can skip this step. See your CICS systems programmer for assistance with these tasks.

1. Upload the `redef.cbl` program to a source data set on the mainframe.
2. Compile the program and place the resulting load module in a data set referenced by the DFHRPL DD statement of the CICS region.
3. Use CEDA to define the REDEF program and install the definition.

Step 2 ? Prepare the file system

During the course of this tutorial, you define models and generate EJB and Web code. This information should be stored in separate directories and mounted for your project.

1. Create a new directory, for example: `\OptimalJ\cobolRedefines`.
2. In this directory, create three subdirectories:

- `\OptimalJ\cobolRedefines\redefinesModels`
- `\OptimalJ\cobolRedefines\redefinesEjbCode`
- `\OptimalJ\cobolRedefines\redefinesWebCode`

Step 3 ? Create new OptimalJ project

Create a new OptimalJ project called `redefines`.

1. Choose **Project>New OptimalJ Project**. Set the project name to `redefines` and click **Next**
2. Select **New Model** as the project type and specify the directory `\OptimalJ\cobolRedefines\redefinesModels` in the **Model dir** field.
3. Enter `redefines` as the **Fully-qualified package name** and choose `Three Tier Application Structure with Integration` as the **Initial Structure**. Click **Next**.
4. Choose **Mount each filesystem yourself** and click **Next**.
5. Click **Finish**. This mounts the `\OptimalJ\cobolRedefines\redefinesModels` directory and creates packages for the domain, application, and integration models there.
6. Mount the directories to contain the generated EJB and Web code (`\OptimalJ\cobolRedefines\redefinesEjbCode` and `\OptimalJ\cobolRedefines\redefinesWebCode`).

   Choose **File>Mount Filesystem**. In the wizard, select **Local Directory**, and select the directory to mount. Click **Finish**.

Step 4 ? Install the resource adaptor

For this tutorial, you need to install the CICS ECI resource adapter in the OptimalJ environment.

1. Choose **Deploy> Install JCA Resource Adapter**
2. In the **Resource Adapter Filename** field, enter the name of the RAR file containing the resource adapter (or use the browse button to locate the RAR file). The ECI resource adapter is located in the file `cicseci.rar` in the `deployable` directory of your IBM CICS Transaction Gateway installation.
3. Click Finish to install the resource adapter. This copies the RAR file to the `resource` directory in your user directory. It also unpacks the RAR file and mounts any jar files within as file systems.

Figure 1-114 Mounted directories and jar files

Step 5 ? Define a CICS ECI connection factory

The JCA component model element created by importing a COBOL program references a connection factory definition. This allows the definition to be used in the generated code and in configuring the EJB server.

1. Choose **Tools>Options** to display the Options panel. Expand the nodes **OptimalJ Configuration>Code Generation>JCA Settings**.
2. Right-click the Connection Factories node and choose **Add ECIResourceAdapter Factory** from the pop-up menu.
3. Enter a name for the **Connection Factory**, for example `cicsecirar`. This is the name of the definition and does not have to match the APPLID of the CICS region. Click OK.
4. Select the connection factory you just created. If not open already, display the properties. The default properties are automatically supplied.
5. Fill in the relevant properties as follows, and close the Options panel:

Table 1-13 Connection factory properties

Property	Description
ConnectionURL	Address of the CICS Transaction Gateway, for example, `tcp://mycompany.com`
PortNumber	Port used by the Gateway (IBM default is 2006)
ServerName	Server name for the CICS region (APPLID). This field is case-sensitive.

Figure 1-115 Connection factory properties for CICS ECI resource adaptor

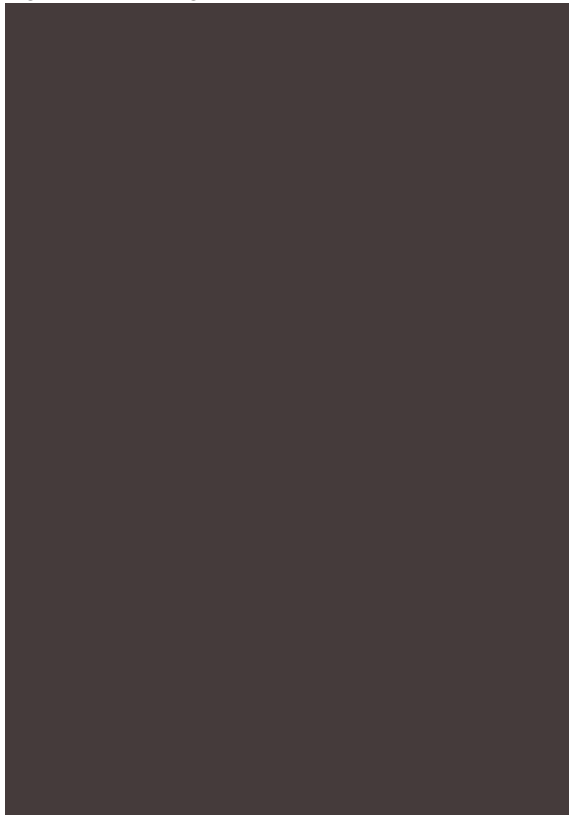
For more information on these settings, see *Define a connection factory*.

Step 6 ? Import the COBOL program

1. Choose **Model>Import Model>Import COBOL from File** to start the COBOL import wizard.
2. Select `integration.jca` and click **Next**.
3. Select Import a CICS Program and click **Next**.
4. Select the `redef.cbl` program. This file is located in a subdirectory of your user directory: `UserDirectory\sampleprojects\integration\jca\cics\redefines\`. The default location of the user directory on Windows is `C:\Documents and Settings\User\.OptimalJ-Edition\3.1`; on UNIX, it is `$HOME`

   In the **JCA Connection Factory** field, select the ECI resource adaptor factory you defined earlier.
5. Click **Finish**. Your tree looks like this:

Figure 1-116 Integration model for JCA



For a description of the integration model elements and how they are mapped from CICS COBOL, see *Structure of JCA integration model (COBOL)*.

Step 7 ? Select the fields of interest for the bean and user interface

In the step 8, you will generate domain and application models from the JCA model. All fields copied to the domain model are copied to the EJB and WEB models, and are ultimately made available in the EJBStructs and JSPs generated from those models. In this step you identify those fields that should *not* be present in the bean and user interface by setting their *isFiller* property to `True`.

To identify those fields *not* of interest in the EJB and WEB models:

1. Select the COBOLField `FORMATTED-DATE` in the JCA model package `redefines.integration.jca`.
2. Locate the *isFiller* property in the Properties pane and set its value to `true`.
3. Repeat this procedure for the COBOLFields `MONTH-CHARS` and `RECORD-DATA`.

In the COBOL program file, the REDEFINES clause is used several times:

```
1 | 05  DATE-CONTENTS REDEFINES FORMATTED-DATE.
2 | ...
3 | 05  MONTH-CHARS REDEFINES MONTH-NAME
4 | ...
5 | 05  RECORD-A REDEFINES RECORD-DATA
6 | ...
7 | 05  RECORD-B REDEFINES RECORD-DATA
8 | ...
```

By setting the *isFiller* property, you determine which field should be used in the bean implementation.

Step 8 ? Generate domain and application models

By generating an application models based on the integration model, you can generate a session bean (and related code) to serve as a wrapper for the JCA component client code and a WEB front end to invoke the session bean.

To generate the domain model and application models from the integration model:

1. Choose **Model>Generate Model>Generate Domain Models>Generate Domain from JCA** to start the wizard.
2. In the wizard, select `redefines.integration.jca` as the source and `redefines.domain` as the target domain model. Accept the default values on the last pane and click **Finish**.
3. Choose **Model>Update All Models** to generate the EJB and Web models from the domain model.
4. In the wizard select `redefines.application` as the destination and click **Finish**.

   Your tree now looks like this:

Figure 1-117 Application models

Step 9 ? Generate and compile code

1. Right-click the `redefines` package and choose **Generate Code**.
2. When prompted for the EJB module code location, choose `\OptimalJ\cobolRedefines\redefinesEjbCode`. When prompted for the Web module code location, choose `\OptimalJ\cobolRedefines\redefinesWebCode`.
3. Choose **Project>Compile Project**to compile the code.

Figure 1-118 Code model

The jca package now contains a `RedefJCAComponent.java` source file and a `recordredef` package directory. The recordredef package contains a `Dfhcommarea.java` class. The RedefJCAComponent class contains code to invoke the JCA interactions defined in the model. The JCA component class can be used as a normal class to invoke the REDEF CICS COBOL program.

In the ejb package, you have a `RedefBean.java` and other files, defining a session bean. The session bean uses the JCA component class in the jca package to invoke the redef CICS COBOL program.

In the web package, you have forms, actions and other files, defining a web front end. The web front end uses the session bean in the ejb package to invoke the REDEF CICS COBOL program.

The EJBStructs and JSPs do *not* contain references to the fields whose *isFiller* property was set to true.

Step 10 ? Edit the generated output JSP

The generated code now contains only a single field for each redefined COBOLField, except for RECORD-A and RECORD-B. The generated JSP must be changed to display the correct record based on the returned value in RECORD-TYPE.

1. In Explorer [Code Model], expand the Web code directory `\OptimalJ\cobolRedefines\redefinesWebCode`.
2. Double-click the file `RedefRedefInvokerOutput.jsp` to open it in the editor.
3. Locate the section that displays the output of `RECORD-A`. This begins as follows:

```
<tr>
<th class="maint-label">
<bean:message key="label.RedefRecordARecordAStringNameA"/>
```

4. Before this code insert the following:

```
<logic:equal value="A" name="redefRedefForm"
     property="dfhcommareaReturnRecordType">
```

5. Locate the section that displays the output of `RECORD-B`. This begins like the following:

```
<tr>
<th class="maint-label">
<bean:message key="label.RedefRecordBRecordBStringHomePhoneB"/>
```

6. Before this code insert the following:

```
</logic:equal>
<logic:equal value="B" name="redefRedefForm"
     property="dfhcommareaReturnRecordType">
```

7. Locate the end of the output display with the `</table>` tag.
8. Before this code insert the following:

```
</logic:equal>
```

9. Press `Control-S` to save your changes.

After excluding the RECORD-TYPE field using the *isFiller* property of the COBOLField and generating the application models and code, there are still two field definitions available?RECORD-A and RECORD-B. By adding logic to the generated JSP, you add the programming required to display the correct record based on the returned value in RECORD-TYPE.

Step 11 ? Test the session bean

You can use the generated Web application to invoke the JCA component via the session bean generated from the EJB model.

1. Choose **Test>Start Application Server**. Ensure that `..\application\ejb\ejb.jar` is selected in the Archive File Selector window and press **OK**. Wait for the server to start. This starts the Web server and browser, displaying the Main Menu. Wait for the server to start. This starts the Web server and browser, displaying the Main menu.
2. Click **Maintenance Redef**.
3. Click **redef**.
4. Enter A in the **functionCode** field and click **OK**.
   The page redisplays with the current date and `RECORD-A` fields.
5. Click **Home**.
6. Click **Maintenance Redef**.
7. Click **redef**.
8. Enter B in the **functionCode** field and click **OK**.
   The page redisplays with the current date and `RECORD-B` fields.

In this tutorial, you have integrated a sample CICS COBOL program containing REDEFINES clauses with an OptimalJ application. It is important to remember that the record framework handles REDEFINES clauses in the same way as COBOL. Each time a redefined field is set, the new values replace the corresponding values in other definitions. Likewise, getting the data of a redefined field uses the same source as other definitions of the field. When a field is defined with both numeric and non-numeric types, care must be used to avoid attempting to get data with a numeric getter when the data in the record may be non-numeric.

Further reading

For more information, see *COBOL REDEFINES clause* and the other documentation on *Integrating with JCA*, as well as the CICS Transaction Gateway documentation.

1.21 Integrating a Web service

This tutorial shows you how to call an external Web service from your OptimalJ application.

The Web service is based on the CRM example. It returns an object that contains the number of calls grouped by their level (normal, critical and enhancement). It requires the customer identifier (custid) as input parameter. The Web service you call in this tutorial is deployed on a Compuware server. The WSDL that is imported into OptimalJ is generated dynamically.

Prerequisites

- You must be familiar with the basic development features of OptimalJ.
- This tutorial assumes you are familiar with the concept of Web services and its related technologies (SOAP, WSDL, XMLSchema).
- The tutorial demonstrates how to call a Web services that is hosted by Compuware. You need to know the values for your proxy host and port, if network communication is routed via a proxy server. The proxy host and port number are required to test the Web service. Ask your system administrator or MIS department for this information.

Note: For this tutorial a database is not required.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial you learn how to call a Web service within your application.

Step 1 - Prepare the filesystem

1. Create a new directory `\wsconsumer`?this directory will hold your model definitions and EJB and Web code in separate subdirectories.
2. Create subdirectories `\wsconsumer\wsModel`, `\wsconsumer\wsEjbCode` and `\wsconsumer\wsWebCode`.

Step 2 - Create a new project

1. From the menu, select **Project>New OptimalJ Project**. Set the project name to `WSConsumer` and click **Next**.

2. Select the type of project. Set the radio group to `new Model`, set the **Model dir** to `\wsconsumer\wsModel` and click **Next**.
3. Set the **Fully-qualified Package Name** to `wsconsumer`. The **Initial Structure** is set to `Three Tier Application Structure with Integration` (default). Click **Next**.
4. Accept the default `Mount each file system yourself` and click **Finish**.

Step 3 - Define a Web service client module

1. In the Explorer [Application Model], right-click `integration.webservices`.
2. Choose **New Child>WSClientModule**.
3. Accept the default name `WSClientModule` and click **Finish**.

Step 4 - Import WSDL file

In this step, you import the Web service description in the OptimalJ integration model. You can import the WSDL file (and related XML Schema files) if you know the URI for it. The WSDL can also be generated dynamically using the Web service classes as source information. In this tutorial you apply the last method.

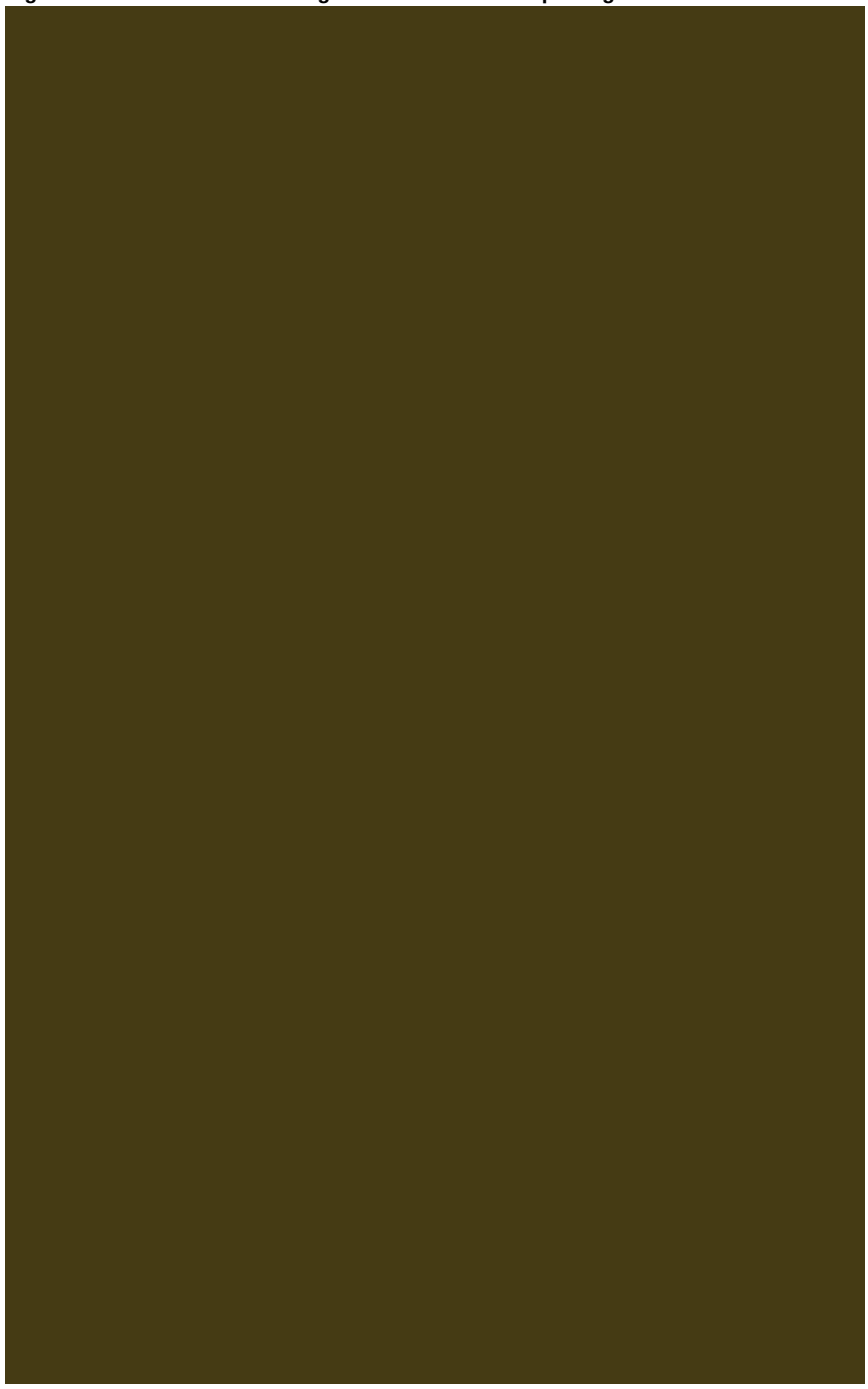
1. If you are working behind a firewall, first check that the proxy host and port are specified. Enter these settings as follows:

- On the menu, choose **Tools>Setup Wizard**. Select **Use Proxy Server** and fill in the **Server name** (IP address) and **Port**.
- Alternatively, on the menu choose **Tools>Options>IDE Configuration>System>System Settings** and assign the proper values to the *Proxy Host*, *Proxy Port*, and *Use Proxy* properties.

2. On the menu. choose **Model>Import Model>Import Web Service WSDL/XMLSchema from File**.
3. In the **WS Absolute URI** field, fill in the following URI `http://javacentral.compuware.com/WebserviceTutorial31/services/CustomerCallOverview?wsdl` and click **Next**.
4. The Web service is protected and requires authentication. You need to provide a username and password. Enter `wstutorial` for the **User Name** and `public` for the password. The WSDL is now dynamically generated and imported in OptimalJ's Web service integration model.

5. Select the integration package `wsconsumer.integration.webservices` to import to and click **Next**.
6. Select the target module `WSClientModule` and click **Finish**.

Figure 1-119 Web service integration model after importing a WSDL file



The import process automatically creates a WSDLRepository called `wsdlrepository` and a WSClientComponent called `CustomerCallOverview`, and a XSDSchema element called `schema` (this is a child element of Types). The schema contains a complex type `CallsOverview`. The WSClientModule is a placeholder for the WSClientComponents; its *part* property refers to a collection of WSClientComponents. The WSClientComponent (`CustomerCallOverview`) is a placeholder for the Web service operations (see the `WSClientComponent.WSConnectorOperation` element).

In this example, the WSClientComponent `CustomerCallOverview` has one operation `getCallsOverview`. The WSConnectorOperation has a property *wsoperation*. The value of this property links to the WSPortType element in the WSDLRepository.

To see the SOAP Address, expand the nodes `wsdlrepository.CustomerCallOverviewService.CustomerCallOverview`. The property *location* of the SOAPAddress element contains the URI.

Step 5 - Set username and password

The Web service host requires a username and password for calling operations on this Web service. You can specify the username and password as properties of the WSClientComponent. The username and password are saved in the deployment information for the session bean that acts as a wrapper for the Web service component.

1. Select the WSClientComponent `CustomerCallOverview`.
2. In the Properties window, enter `wstutorial` for the *userName* and `public` for the *userPassword*.

Step 6 - Define Web service security properties

The SOAP request message that is sent to the provider of the Web service needs to be secured. The algorithms used for generating a message digest and a digital signature, are determined by the properties of the `WSSigningProperties` model element and are agreed on with the receiver of the message, who validates the message.

1. Right-click the `webservices` package and choose **New Child>WSSigningProperties**.
2. Enter `signer` in the **Name** field.
3. Enter the following property values for the signer element (some of them are default):

Table 1-14 signing properties

Property	Value
canonicalizationMethod	`C14N Exclusive Omit Comments`
keyInfoMethod	`X509Certificate in X509Data`
mustUnderstand	`True`
signatureMethod	`DSA-SHA1`
transformMethod	`C14N Exclusive Omit Comments`

For a detailed explanation of these properties, see *Defining signing properties*.

Note: the message receiver rejects the message if you select different values!)

4. Select the WSClientComponent `CustomerCallOverview` to assign the security properties to the Web service component.
5. Click the property *securityProperties*. Click the Browse button and add the `signer` object to the property.

Step 7 - Generate domain and application models

By generating an EJB model based on the integration model, you generate a session bean (and related code) to serve as a wrapper for the Web service client code. Although you can generate the EJB model directly from the integration model, it is better to first generate the domain model. This enables you to generate a Web model that you can use to create a user interface for your Web service operation.

1. On the menu choose **Model>Generate Model>Generate Domain Models>Generate Domain from Web Service** to start the wizard.
2. In the wizard:

- Select `Call Out` and click **Next**.

- Select the `webservices` package in the integration model and click **Next**.
- Select `wsconsumer.domain` as the target model and click **Finish**.

Note: the domain class model contains a package `schema` with a DomainStructType `CallsOverview`. The domain service model contains a domain service `CustomerCallOverview` with an operation `getCallsOverview`.

3. On the menu choose **Model>Generate Model>Generate Application Models>Generate EJB from Domain**. In the wizard:

- Select `wsconsumer.domain` as the source
- Select `wsconsumer.application.ejb` as the target and click **Next**.
- Accept the default generation options and click **Finish**.

4. On the menu choose **Model>Generate Model>Generate Application Models>Generate WEB (EJB based) from Domain**. In the wizard:

- Select `wsconsumer.domain` as the source
- Select `wsconsumer.application.web` as the target and click **Finish**.
- Accept the default generation options and click **Finish**.

Figure 1-120 Application model after generating models

The domain class model now contains a domain struct type (CallsOverview) with three fields: normal, critical, and enhancement. This struct is the type of the return parameter of the getCallsOverview operation. The domain service model contains a domain service CustomerCallOverview with one operation (getCallsOverview).

The EJB Model contains a EJB session component (CustomerCallOverview).

The Web model has a Web component (again called CustomerCallOverview). The Web action getCallsOverview invokes the business method with the same name on the EJB session component.

*Note: Instead of generating the EJB and Web models individually, you can choose **Model>Update All Models**. This option additionally generates the DBMS model, which is not used in this tutorial.*

Step 8 - Generate code

After generating the EJB and Web models, you can generate the application code. The code generated for the Web services integration model includes client stub code and helper classes used to call out to the Web service.

1. On the menu choose **Model>Generate All Code**.
2. You are prompted to select a filesystem for the code of the ejb module. Click **Mount New Filesystem** and select the folder `\wsconsumer\wsEjbCode`. Click **Ok**.
3. Select a filesystem for the code of the Web module. Click **Mount New Filesystem** and select the folder `\wsconsumer\wsWebCode`. Click **Ok**.

Step 9 - Modify the client-config.wsdd file

When generating the code for the EJB module a configuration file is created called `client-config.wsdd`. This file is used by the Axis toolkit and includes information about the keystore used. This information needs to be provided by you.

1. In the Explorer[Application Model] select the `application.ejb` package.
2. Right-click the EJB module `ejb` and choose **Edit Generated Files>client-config.wsdd**.
3. Enter the following values for the attributes as available in the free block:

```
 1 |  <parameter name="keystoreFile"
value="OptimalJInstallDir\docs\tutorial\mykeystore"/>
 2 |         <parameter name="keystorePass" value="changeit"/>
 3 |         <parameter name="privateKeyAlias" value="example"/>
 4 |         <parameter name="privateKeyPass" value="changeit"/>
 5 |         <parameter name="certificateAlias" value="example"/>
```

```
6 |         <parameter name="keystoreType" value="jks"/>
```

The keystore file `mykeystore` is located in the OptimalJ installation directory (subdirectory `docs\tutorial`). This file is generated with the keytool utility, available in your JDK. The file has been created with the following command:

```
keytool
-genkey
-dname "cn=OptimalJ, ou=Tutorial, o=Compuware, c=US"
-alias example
-keypass changeit
-keystore C:\OptimalJInstallDir\docs\tutorial\mykeystore
-storepass changeit
-validity 400
```

This command creates a keystore named `mykeystore` in the directory `c:\optimalj`, and assigns it the password `changeit`. It generates a public/private key pair for the entity whose "distinguished name" has a common name of `OptimalJ`, organizational unit of `Tutorial`, organization of `Compuware` and a two-letter country code `US`. The default keystore type is `jks`. It uses the default DSA key algorithm to create the keys.

For more information on the keytool program, see *http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html*

Step 10 - Specify EJB execution settings

The CustomerCallOverview Web service is hosted by Compuware. Because this Web service is outside your company's environment, you need to specify some additional settings if communication is channeled via a proxy.

1. To specify the proxy settings, on the menu choose **Tools>Options>Debugging and Executing>Execution Types>EJB Execution**.
2. Use the property *Java properties*, to specify your proxy host name and port number. Type: -`Dhttp.proxyHost=`*`ProxyHostNameOrIPAddress`* and -`Dhttp.proxyPort=`*`ProxyPortNumber`*. (Separate the two settings with a space.)

These setting are needed by the application server at run time to call the Web service and obtain a reference to the Web service endpoint.

*Note: If you want to start the application server in debugger mode, you also need to add these settings under **EJB Execution Debugger**. If the Web service needs to be accessed via the HTTPS protocol, the EJB Execution properties need to change accordingly: `-Dhttps.proxyHost=ProxyHostNameOrIPAddress` and `-Dhttps.proxyPort=ProxyPortNumber`. If you change the Java properties, the application server must be restarted.*

Step 11 - Call the Web service

You are now ready to call the Web service. In this tutorial, a Web interface is used to call the Web service operation. This Web interface is generated from the Web model.

1. On the menu, choose **Project>Compile Project** to compile all code.
2. On the menu choose **Test>Start Application Server**. Wait for the server to start.
3. Start the logging server by choosing **Test>Start Logging Server**. Its output window contains information concerning the SOAP message processing which is interesting to read.

Note: This step is not mandatory.

4. Click *Maintenance CustomerCallOverview*, and then on the toolbar *getCallsOverview*.
5. Call the Web service, by filling in a value for the input parameter. Existing values are: 0001, 0002, 0003 and 0004. Click **OK** to see the return values.
6. In the output window of the logging server, search for the SOAP message that has been sent to the Web service host.

The field label for the input parameter might be something like **in0**. This is because when importing the Web service, the WSDL is generated dynamically. If, in this process, a part name in the WSDL can not be determined, a default name is used for it.

If the returned values are all -1, the submitted customer id does not exist. If there is a connection failure, check your proxy settings (as specified in Java properties under **Tools>Options>Debugging and Executing>Execution Types>EJB Execution**). If you receive a security related error, check the correctness of the values of the signer element and the settings in the `client-config.wsdd` file. Although far less likely, it can also indicate that the Web service itself is not running.

In this tutorial you learned how to integrate an external Web service into your OptimalJ application. The process starts with importing a Web service. The Web service interface description is generated dynamically (derived from the Web service classes, as demonstrated in this tutorial), or is imported from a existing WSDL file. The next steps concern the generation of domain and application models. The generated session component in the EJB model acts as a wrapper for the Web service client component in the integration model. The generated Web component can be used as a simple user interface to test the Web service call-out. In most cases, proxy settings need to be set to establish a connection between the Web service client component and the Web service on the provider's host machine.

Further reading

For more information, see *Consuming a Web service*.

1.22 Developing and deploying a Web service

This tutorial shows you how to create a Web service as part of the default CRM application. This Web service can subsequently be called by other applications to retrieve information about a customer's service level. The Web service contains one operation, getServiceLevel. The input parameter is the customer ID; the return value is its service level (BRONZE, SILVER, or GOLD).

Prerequisites

- You must be familiar with the basic development features of OptimalJ.

  This tutorial also assumes that you followed the tutorial *Your first OptimalJ application*.
- This tutorial assumes you are familiar with the concept of Web services and its related technologies (SOAP, WSDL, XMLSchema).

Duration

This tutorial takes approximately 45 minutes to complete.

Objectives

In this tutorial you will learn to develop a Web service that exposes functionality of the CRM application over the Internet.

Step 1 - Prepare the filesystem

If you previously created OptimalJ projects using the default CRM example, you have to create a new directory for the example application.

- Create a new directory `\wsprovider`?this directory holds your model definitions as well as the generated code for the Web, EJB and Web service integration models. You can decide where to create this directory, although it is recommended not to do this in your `OptimalJUserdirectory`.

Step 2 - Create a new project containing the CRM example

The CRM example is a sample application delivered with OptimalJ that demonstrates features and functionality available in OptimalJ. You can enable the CRM application when creating a new OptimalJ project.

To create a new project based on CRM:

1. From the menu, select **Project>New OptimalJ Project**. Set the project name to `WSProvider` and click **Next**.
2. Select the type of project. Set the radio group to `Experiment with one or more example Models` and set the **Unpack dir** to `\wsprovider`. Click **Next**.
3. Click the CRM Example Module check box *on* and click **Next**.
4. Click **Finish**.
5. Choose **Model>Generate Model>Generate Domain Models>Generate Service from Class**.

Figure 1-121 The CRM domain class and service models

Step 3 - Define a domain service operation

The domain service `CustomerSvc` is extended with the operation `getServiceLevel`. This domain service will be transformed into a Web service in subsequent steps. The domain service `CustomerSvc` uses a domain view with Customer as root class. The classes `Call` and `ServiceAgreement` are included (byValue).

Note: You may want to create a new model package in the service model, for example, mywebservices. This package than acts as a placeholder for all domain services which, at the end, are deployed as Web services. In this tutorial, the default package structure is used.

1. In the Explorer [Domain Model], expand the nodes `crm.domain.service`.

   Right-click the domain service `CustomerSvc`, and choose **New Child>DomainServiceOperation** to start the Edit DomainService wizard.

2. Click the top Add button to add a domain service operation. In the **Name** field, enter `getServiceLevel` and select **Return Type** `String`.

   Click the bottom Add button to add a parameter and give it the **Name** `custid`, **Type** `String`, **Kind** `in`.

Figure 1-122 Edit DomainService

3. Click **Finish**. The expanded domain service model looks as follows:

Figure 1-123 Domain service model

Step 4 - Generate models

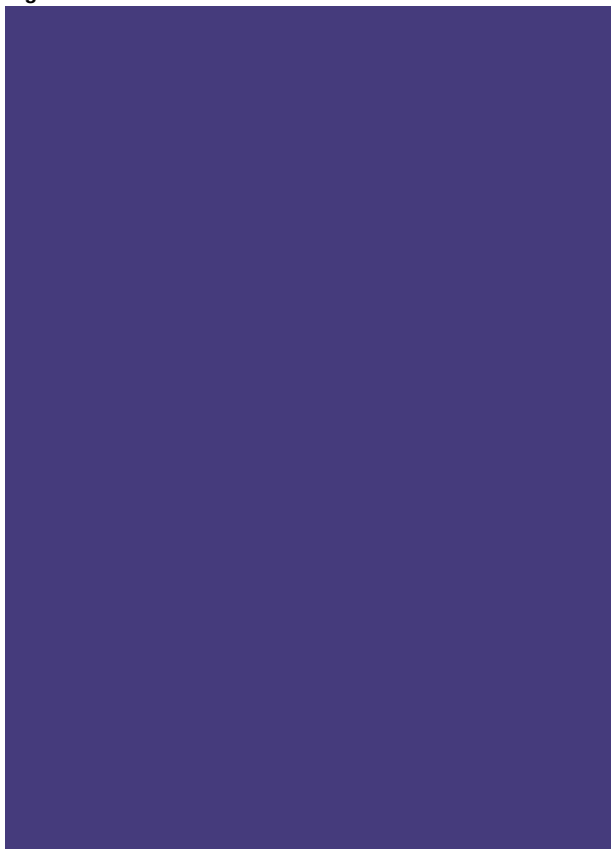
You can now generate the application models from the domain, and from there the integration model from the EJB application model.

1. Choose **Model>Update All Models**.

Note: This menu option also generates a Web model that can be useful for testing, as you will see later. Strictly speaking it is sufficient to generate the EJB model and the DBMS model. The DBMS model is needed if the Web service implementation requires access to the classes in the domain class model.

2. In the wizard, select the `crm.application` model package and click **Finish**. This generates the application models.

Figure 1-124 Generated EJB model



3. Choose **Model>Generate Model>Generate Integration Models>Generate Web Service from EJB**.
4. In the wizard:

- Select `crm.application.ejb` as the source and click **Next**.
- Select `crm.integration.webservices` as the target. Click **Next**.
- Click **Finish**.

Figure 1-125 Generated Web services integration model

5. In the Explorer [Application Model], expand the `application.web` nodes and select the module `web`.
6. In the properties sheet, click the *usedModule* property and use the Browse button to examine the value. Note the Web service server module `crm.integration.webservices.ejbWSServerModule`.

The Web service will run in the context of a Web application. The Web module element represents this Web application. This is the reason why the WSServerModule in the integration tier should be assigned to the *usedModule* property of the Web module, before the code is generated.

The WSServerComponent element `CustomerSvc` in the `integration.webservices` package, represents the Web service. Notice that no WSServerComponent is generated for the other EJBSessionComponent `ServiceAgreementSvc`. This component has no business methods and therefore, it doesn't make sense to expose it as a Web service.

Step 5 - Enable basic authentication

The consumer of your Web service needs to provide a user name and password to be able to use the Web service. This step shows how to enable authentication.

Note: authentication is one of the security requirements. Verifying the integrity of the received SOAP message and the validation of the identity of the sender requires the use of digital signatures and certificates. The implementation of these security requirements are not included in this tutorial. The tutorial Integrating a Web service demonstrates how the Web service consumer signs a SOAP message. The topic Verifying SOAP messages provides the information for setting verifying properties within OptimalJ.

1. In the Explorer[Application Model], right-click the `application.web` package and choose **New Child>WEBSecurityRole**.
2. Enter `role1` in the **Name** field and Click **Finish**.
3. Select the `integration.webservices` package.
4. Right-click the WSServerModule `ejbWSServerModule` and choose **Edit**. Click **Next**.
5. Enable the security role `role1`. Click **Finish**.

`Role1` is a role default available the `tomcat-users.xml` file. The user name and password associated with this role are `role1` and `tomcat`. The `web.xml` file, which is generated as part of the next step, will contain security and login constraint information in the `<security-constraint>` and `<login-config>` tags. Additionally, the Axis configuration file `server-config.wsdd` contains security information to inform the Axis engine which operations are allowed to be called.

Step 6 - Generate code, WSDL and XMLSchema files

Now that the application and integration models are defined, you can generate and modify the code. Generating code also generates a WSDL file for the Web service and a XMLSchema file containing the type definitions. Security information (see previous step) is included in the `web.xml` and `server-config.wsdd` files.

1. Choose **Model>Generate All Code**.
2. If you are prompted for the location of generated code for the ejb module, select `\wsprovider\crmEjbCode` and click Ok.

3. If you are prompted for the location of generated code for the web module, select `\wsprovider\crmWebCode` and click Ok.

The Web service is generated from the `ejb` model package. The name of this package is used in the name of the WSDL file, `ejb_top.wsdl`, the name of the directory `ejbWSServerModule` and the XMLSchema file `ejbxsdschema`.

The `ejb_top.wsdl` file is also copied to the `\wsprovider\crmWebCode` directory. The directory `ejbWSServerModule` contains some code specific for Web service integration and is located in the folder `\wsproviderModel\crmexample\crm\integration\webservices`

The `server-config.wsdd` file located in `\wsproviderModel\crmWebCode\WEB-INF`.

Step 7 - Implement Web service logic and compile all code

You now need to implement the Web service logic in the code generated in the previous step. The implementation provided here is one solution, alternatives are possible. The code is simplified a little bit.

1. In the Explorer [Application Model], right-click the EJBSessionComponent `CustomerSvc` in the `ejb` package.
2. Choose **Edit Free Blocks in Generated Files>BusinessMethods>CustomerSvcBean.java>body(getServiceLevel**
3. In the Source Editor, enter or copy the following code in the free block:

Example: Implementation of getServiceLevel method

```
1 |      returnValue = "No customer found";
2 |         try {
3 |         crm.application.ejb.customersvc.CustomerDataObjectCollection
  |           c = getAllFindByProfileOnKey(custid);
4 |         Iterator it = c.iterator();
5 |         while (it.hasNext()) {
6 |             crm.application.ejb.customersvc.CustomerDataObject
  |               cdao =
(crm.application.ejb.customersvc.CustomerDataObject)
  |               it.next();
7 |             crm.application.ejb.customersvc.ServiceAgreementDataObject
```

```
   |              sadao =
cdao.getServiceAgreementServiceAgreementDataObject();
 8 |            returnValue = sadao.getServiceLevel().toString();
 9 |            };
10 |         } catch (Exception e) {};
```

4. Choose **File>Save** to save the code change.
5. Choose **Project>Compile Project** to compile all code.

The method `getAllByFindProfileOnKey` returns all CustomerDataObjects whose key match the profile. In our case only one customer matches, so the collection contains one CustomerDataObject. The method `getServiceAgreementServiceAgreement` returns the ServiceAgreement that is associated with the Customer (there is a 1:n relationship between ServiceAgreement and Customer). The `getServiceLevel` method returns an enumeration type, which is converted to a String by the `toString` method.

Step 8 -Test the Web service

The code generated for the Web tier includes two Java Server Pages that provide a simple user interface for the `getServiceLevel` operation. In this step you use this interface to test the internal logic of the Web service operation. Additionally you can verify if the generated WSDL file is protected by a username and password.

Note: This step only tests the internal implementation of the Web service.

To quickly test the implementation of your Web service:

1. Start the default SOLID database.
2. Choose **Test>Start Application Server**. Wait for the server to start.
3. Navigate to the getServiceLevel link:

- Click Maintenance CustomerSvc.
- Click Browse.
- Click one of the Edit buttons.
- On the toolbar at the top, click the getServiceLevel link.

  On the input screen you can fill in a customer id. Valid values are 0001, 0002, 0003 and 0004. Click on Ok to get the return value.

4. Start a new browser session and enter `http://localhost:8081/services` in the **Address** field.

5. A dialog window prompts you for a user name and password. Enter `role1`, for the **User Name** and `tomcat` for the **Password**. You can now view the wsdl for the Web service created in this tutorial.

Step 9 - Deploy the Web service

The Web service is now ready to be published. You may need to add application server specific deployment information. You also need to provide the URL of the Web application that hosts your Web service. The WSDL and XMLSchema files are packaged in the `web.war` file.

The tutorial *Integrating a Web service* demonstrates how to call a Web service.

In this tutorial you learned how to develop a Web service. The process of developing a Web service is very similar as developing other components. You start in the domain model with creating a domain service and, if required, adding complex types in the class model. After that you generate the EJB and Web models. The Web service component is generated from the EJB model. In the integration model, you can add security requirements for authentication and digital signatures. You can use the Web user interface as a simple interface for internal testing of your Web service logic. Also, when generating code for the Web module, the file `server-config.wsdd` is created in the `WEB-INF` folder (but only, if security features are added to the Web service).

Further reading

For more information, see the documentation on *Providing a Web service* and *Web service security*.

1.23 Creating the application EAR

OptimalJ creates applications in accordance with the applicable J2EE, Java, and related standards. Consequently, you can deploy your OptimalJ applications on a wide range of application servers and databases.

To deploy the application you need to create an Enterprise ARchive file (EAR file), which contains all the information an application server needs to run the application. You can then deploy the EAR file using the deployment tools supplied with your application server. After moving the EAR file to the target platform, you can add it to the application server?s configuration, and deploy the application quickly and easily. OptimalJ?s *Assembly Workbench* utility helps you create EAR files.

Prerequisites

- This tutorial assumes that you previously followed the tutorial *Importing a domain model*. If you did not follow this tutorial, you can use another application that you have created and tested with OptimalJ.
- You must be familiar with the basic development features of OptimalJ.

Duration

This tutorial takes less than half an hour to complete.

Objectives

This tutorial shows you the steps needed to create an Enterprise ARchive file (EAR file). The EAR file contains JAR, WAR and other archives (like the optional DAR file), needed to deploy an enterprise application. You can then deploy the EAR file using the deployment tools supplied with your application server.

Step 1 ? Explore the application archive files

Enterprise archive files require several modules from your application. This step shows you where the individual archives are located in your application. The `.jar`, `.war` and `.dar` files are needed to create the enterprise archive file that you are going to deploy.

1. From the menu, select **Project>Project Manager**.

2. Select the project that contains the application created when following the tutorial *Importing a domain model*. Click Open.

Note: The package and path names in this tutorial assume that you're using the project from the Importing a domain model tutorial. If you are using a different project for this tutorial, substitute the appropriate package and path names.

3. Go to Explorer [Code Model].
4. Expand the nodes `modelpackage.application.dbms`.
   The dbms package contains a Database Archive file with extension `.dar`.

Note: The DAR file is specific to OptimalJ. This file is used to create the database tables in the OptimalJ test environment. This file is not intended for use in deployment environments external to OptimalJ. For other deployment environments, such as Sun? ONE Application Server (formerly iPlanet), the database tables need to be created using the OptimalJ SQL Workbench, or other database-specific tools.

5. Expand the nodes `modelpackage.application.ejb`.
   The ejb package contains a Java Archive file called `ejb`. By default, the Explorer [Code Model] does not display the `.jar` extension for this file.
6. Expand the nodes `modelpackage.application.web`.
   The Web package contains a Web ARchive file with extension `.war`.

Step 2 ? Create the EAR definition file

The contents of the EAR file are stored in an EAR definition file. The EAR definition file also specifies if the `ejb.jar` file included in the EAR file should be customized for a specific application server.

1. From the menu, select **Deploy>Start Assembly Workbench**.
2. In the **File Name** field, enter the name of the EAR file you want to create, for example, `MyDeploy`.
3. Use the center pane of the Select/create Enterprise Archive wizard to select the directory in which to create the file.
   Select `\modelpackage`.
4. Click Create.

Go to the Explorer[Code model] tab. The `MyDeploy` EAR definition file is added to `\modelpackage` directory.

5. The Source Editor window of Assembly Workbench allows you to edit the EAR file.

   The MyDeploy tab (matches the name of your EAR definition file) contains the high-level options for the EAR file, such as icons and descriptions.

   Some versions of certain application servers require special customizations to the `ejb.jar` file that is packaged in an EAR file. If you see your application server listed on this tab, click the corresponding button. When you compile the EAR definition file, OptimalJ will perform any required customizations to the `ejb.jar` file, corresponding to the application servers you specified. The customized `ejb.jar` is packaged in the EAR file.

Note: Select the WebSphere button for all supported WebSphere versions, not just 5.

Figure 1-126 EAR definition settings

6. Click the Modules tab. Click Add Module to add a module to the archive.
7. Click the `File Name` field, and then click the Browse button.
8. Browse to `\modelpackage\application\ejb` and select the file `ejb.jar`.

Note: Two-tier applications that do not have an `ejb.jar` file and can skip this step.

9. Add another module to the archive. Click Add Module, click the browse button for the `File Name` field and select the `web.war` file located in `\modelpackage\application\web`.

   When the `web.war` module is selected, the **Context root** field is available to set the context root of the deployed application.

Figure 1-127 MyDeploy EAR definition file with two modules

Note: The module types are updated automatically.

10. Click the OtherFiles tab.
11. Click Add OtherFile.
12. Click the **Other file** field, and then click the Browse button.

13. Browse to `\modelpackage\application\dbms` and select the file `modelpackage.dar`.
14. In the **Description** field enter `dbms`.

Note: The `.dar` file needs to be included as 'other files' because it is specific to OptimalJ; other deployment environments do not use this file.

15. The EJB module, `ejb.jar`, is dependent on the `alturalibDeployEJB.jar` that is included in your OptimalJ installation. To accommodate this dependency, you must add the `OptimalJ_Installation/libDeploy/alturalibDeployEJB.jar` to your EAR definition.

    Click Add OtherFile to add a the `alturalibDeployEJB.jar` file to the archive.

Note: Two-tier applications that do not have an `ejb.jar` file and can skip this step.

16. Click the **Other file** field, and then click the Browse button.
17. Browse to `OptimalJ_Installation/libDeploy`, where *OptimalJ_Installation* is the location of your OptimalJ installation, and select the file `alturalibDeployEJB.jar`.
18. In the **Description** field enter `Deploy EJB`.
19. Close the MyDeploy Assembly Workbench window, and click Save when prompted.

Step 3 ? Compile the EAR file

Once the EAR definition file is created, you can generate the EAR file at any time by compiling the EAR definition file. When you perform a compile or build operation on a directory that contains the EAR definition file, OptimalJ compiles the EAR definition file as well.

Note: Depending on the location of your EAR definition file relative to your EJB and Web archives, OptimalJ may compile the EAR definition file before it compiles the `ejb.jar` and `web.war` files. As a recommended practice, always manually recompile the EAR definition file after you perform a compile or build on you project.

1. Before using the `MyDeploy` EAR Definition file, you need to compile it into an `.ear` file.

   In the Explorer[Code model] window, right-click `MyDeploy` and select **Compile**. The `MyDeploy.ear` file is added to `\modelpackage` directory.

Note: During EAR compilation, OptimalJ may customize the `ejb.jar` and `web.war` archives to match your chosen application server (for example, WebSphere). If OptimalJ's integrated test environment is configured to use a different application server (for example, JBoss), it may not be able to use the customized archives. Delete the `ejb.jar` and `web.war` archives and recompile your project before testing your project in the integrated test environment.

This tutorial guided you through the steps required to package an OptimalJ application. The tutorial steps can vary when, for example, you add your own modules in the Assembly Workbench, or when you edit the deployment descriptors of your application. You have created an EAR file that can be deployed on the default application server. To deploy an EAR on another J2EE-compliant application servers, additionally you need to create the server-specific deployment descriptors. For more information about how to create deployment information for various application servers, see *Generating EJB deployment descriptors*.

At runtime, all OptimalJ applications require classes in the `OptimalJ_Installation/libDeploy/alturalibDeployApplication.jar` file. When deploying your application to your application server, you should add this file to the classpath of the application server. This step is covered in the deployment procedures for each specific application server.

Refer to the other topics on deployment for procedures for specific application servers.

Further reading

For more information, see the documentation on *Deploying an application*.

1.24 Creating your own technical key generator

OptimalJ's technology patterns generate technical keys at application model level, when the domain classes do not have a *primary* domain unique constraints. The code generated from the application models implements a Universal Unique Identifier (UUI) as a 32 digit string for the technical key.

OptimalJ's implementation of the technical key is configured through a setting in the `FactorySettings.properties` file. The technical key generator implements the *PrimaryKeyGenerator* interface and uses the *singleton* pattern to improve performance by having only one *key generator* object.

You can implement your own technical key generator to create unique values that identify an instance of a class.

Prerequisites

- You are familiar with the basic development features of OptimalJ and have followed the tutorial *Creating a two-tier application (DAO component)*.

Duration

This tutorial takes approximately 30 minutes to complete.

Objectives

This tutorial extends the application generated in the tutorial *Creating a two-tier application (DAO component)*, so you first import the application and then you implement your own technical key generator class. Finally you make the key fields visible on your JSPs to visualize the values.

Step 1 - Prepare the filesystem

Location for the project files.

In your file system, create the following directory:

1. `\OptimalJ\technicalkey`?this directory will hold the imported application.

In this tutorial, you only create one directory because the application export file recreates the model structure and the code under the given folder.

Step 2 - Create a new project

It is convenient to perform each tutorial in a new project.

To create a new project use the Project Manager wizard.

1. On the menu, select **Project>Project Manager** and click **New**.
2. Set the project name to `technicalkey` and Click **OK**.

Note: You create an empty project.

3. In the Explorer [Code Model], right-click `Code Model` and choose **Mount>Archive files**.
4. Browse to *`OptimalJ Installation directory`*`\src`, select `alturalib-src.zip` and click **Finish**.

Note: `alturalib-src.zip` contains code sources for the key generator and configuration files.

This way you created an empty project for the application.

Your *technicalkey* project contains all the OptimalJ environment except the mounted folders.

Step 3 - Import the application

The two-tier application is based on CRM using DAO components. The application was created by importing from UML in a two-tier structure (Web and DBMS) supporting access to the data through DAO.

An application export contains the results of the tutorial *Creating a two-tier application (DAO component)*.

Import the application.

1. On the menu, select **Model>Import Application**. Browse to `OptimalJ installation directory\docs\tutorial\crmtwotier.zip`.
2. Select the destination directory. Enter `\OptimalJ\technicalkey` as the directory in which to import the application.

This tutorial focuses on the changes in the code level for the technical key implementation, therefore an application import is your starting point to extending the code model.

Another option is to start from the project *TwoTierApplication*, if you have performed the tutorial.

You can also choose to do your own implementation in another project.

Step 4 - Create your own java implementation for the technical key

The application uses a technical key generator created by OptimalJ implementation patterns. You can find the `BasicPrimaryKeyGenerator.java` in `alturalib-src.zip`.

Create the `TechnicalKeyGenerator.java` to implement the *PrimaryKeyGenerator* interface, and then change the **KeyGenerator** factory setting.

1. In the Explorer [Code Model], navigate to `crmtwotier.model.application`, right-click `application` and choose **New>Java Class**.
2. In the New Wizard - Java Class wizard, enter `TechKeyGenerator` as the **name** of the object. Click **Finish**.
3. In the Source Editor window, add the code to implement the *PrimaryKeyGenerator* interface. Replace the code of the file with the following example.

```
 1 | package crmtwotier.application;
 2 | import com.compuware.alturadev.application.PrimaryKeyGenerator;
 3 | /*
 4 |  *
   |    TechKeyGenerator.java
 5 |  *
 6 |  * Very simple keygenerator for demonstration purposes
 7 |  *
   |    Beware that since the techkeygenerator never saves it
   |    state
 8 |  * it will generate the same number whenever it is restarted
 9 |  */
10 | public
   |   class TechKeyGenerator implements PrimaryKeyGenerator {
11 |
12 |     private
   |       static PrimaryKeyGenerator singleton = new TechKeyGenerator();
13 |     private
   |       static int counter = 0;
```

```
14 |
15 |     /**
16 |      * This method returns a singleton reference to the
   |        one and only simpleprimarykey generator
17 |      * (this method is used by the ConfigurableFactory)
18 |      *
   |        @return the singleton implementation of the primary
   |        key generator
19 |      */
20 |     public static PrimaryKeyGenerator getInstance() {
21 |         return singleton;
22 |     }
23 |
24 |     public String getUniqueKey(Object sourceObject) {
25 |         StringBuffer key = new StringBuffer("AutoTechKey");
26 |         counter++;
27 |         if
   |           (counter<100) key.append("0");
28 |         if (counter< 10) key.append("0");
29 |         key.append(counter);
30 |         return key.toString();
31 |     }
32 | }
```

4. On the menu, select **File>Save**.
5. Right-click the Source Editor and choose **Compile**.

Notice the `getInstance()` method, the technical key generator uses a singleton.

Step 5 - Include your technical key generator in the module

To use your technical key generator in the application, you need to included it in the deployable units (war, jar) files.

For this application, there is only one deployable unit, the **web.war**.

1. In the Explorer [Application Model], navigate to`crmtwotier.application.web` and select the web module.
2. In the Properties window, select the *containedFiles*, and click **Browse**.

3. Click **Add** and enter `/crmtwotier/application/TechKeyGenerator.class`.

Note: The separator is the forward slash. Also used at the beginning to define the root package.

4. Click **OK** twice.

By adding the file to the *containedFiles* property, the class is included in the WAR file.

Step 6 - Return your file in the factory for the key generator

The code defines a factory object configurable in the properties file.

1. In the Explorer [Code Model], select the folder `\OptimalJ\technicalkey\webcode`, browse to the `WEB-INF\Classes\com\compuware\alturadev\application`.
2. Open the `FactorySettings.properties`, select the **KeyGenerator**.
3. Enter `crmtwotier.application.TechKeyGenerator` for the Value of **KeyGenerator**.

Note: In the properties file, the separator is the dot.

4. From the menu, select **File>Save** to save the changes.

By changing the **FactorySetting.properties** you specified the class to use for the technical key generator.

Step 7 - Make the uniqueId attributes visible

In order to view the values of the technical key generator, make the **uniqueId** attributes visible in the Web pages.

Technical key do not have a meaningful value for the business, so they are hidden.

1. In the Explorer [Application Model], navigate to `crmtwotier.application.web.serviceagreement.ServiceAgreement.uniqueId` web data attribute.
2. In the properties windows, select the *readAllowed* property and change to `True` by double-clicking the property.

3. Select the *webtype* property and change the value from `DataTypes.webDataTypes.HTMLInputHidden` to `DataTypes.webDataTypes.String`.
4. Repeat the above steps for all **uniqueId** fields you want to see in the JSP.
5. Navigate to `crmtwotier.application.web`, right-click web and choose **Generate Code**.

Note: Since you only modified Web model elements, you do not need to generate all the code, generating the code for the Web model is efficient.

6. In the Explorer [CodeModel], right-click `crmtwotier` and choose **Build All**.

Technical keys are per definition technical implementations of uniqueness for the instances, so they do not have a meaning for the application user. OptimalJ's transformation patterns generate the corresponding attributes in the web data classes as hidden types.

You change the presentation type of the attribute as well as is accessor property to enable the JSP to show it.

Additionally, you generated code only for a model using the pop-up menu. This option exclusively generates code for the hierarchy under the model element, it does not invoke a model checker.

Finally, you used the **Build All** option to compile the project. The build option first removes compiled files then compile the sources living you a *clean* build.

Step 8 - Test the application

Test the application by navigating and adding new records.

You need to start the environment where your application executes.

1. From the menu **Test>Start Application Server**.
2. In the browser, click **ServiceAgreement**.
3. Click **New**. Note the **uniqueId** field is visible and displays the value `AutoTechKey001`

Note: The `AutoTechKey001` is the first value defined in your technical key generator implementation.

You implemented a singleton in your technical key generator, but your class does not save its state. Therefore every time you start the server the number generated will be `AutoTechKey001`.

On top of an imported application, you created your own implementation of the technical key generator by implementing the *PrimaryKeyGenerator* interface, and using the *singleton* pattern. Then you configured the factory `KeyGenerator` to use your class.

Finally, to visualize the values of the technical key `uniqueId` on the JSP, you changed data type and the access property of the web attributes.

Further reading

For more information, see also *Generation of technical keys* under the code model.

1.25 Creating Implementation Patterns

Implementation patterns enable you to define the code generated from an application model. You can influence or replace the code generated by OptimalJ's default implementation patterns!

In this tutorial you create an implementation pattern called *WebHelper* that generates utility classes for WEB components - so this implementation pattern works with the default WEB implementation pattern provided with OptimalJ. The utility classes provide a framework for logging and encrypting data. Instead of handling the logging and encryption during this tutorial, the pattern only creates an empty class with free blocks for logging and encryption code.

Prerequisites

This tutorial is intended for developers and architects starting to use OptimalJ's implementation pattern features.

You must have a strong grasp of the OptimalJ application and domain models, and a good understanding of the tools used to develop implementation patterns.

You should be familiar with the MDA architecture, and the role played by implementation patterns in this architecture.

Above all, you must have considerable knowledge of the target application type and implementation language. For example, to create an implementation pattern for applications using EJBs, you must be an expert in the design and construction of EJBs. The target application type in this tutorial is J2SE.

Duration

This tutorial takes approximately half an hour to complete.

Objectives

At the end of this tutorial you understand the essential concepts and tasks for developing implementation patterns, and have enough knowledge to continue learning about implementation pattern development independently.

You use the following features of OptimalJ in this tutorial:

- Explorer [Meta Model]?browses through the metamodels that define the features of the domain and application models.
- Explorer [Transformation Model]?houses your transformation models.

  Transformation models define the high level structure of your implementation pattern.
- *Code generation framework*?OptimalJ creates a code generator for your implementation pattern from the transformation model. The code generation framework has built-in features for free and guarded blocks, model-to-code navigation, management of generated files, inter-pattern collaboration, and so on.

  You can build the code generator using TPL, Java, or a combination of these two languages.
- *TPL language*?TPL is a easy-to-learn language that enables you to construct your implementation pattern declaratively. TPL hides the details of interacting with the code generation framework behind simple statements, freeing you to concentrate on developing your pattern.
- *TPL editor and compiler*?the TPL editor provides syntax highlighting and guarded block support, so you are protected from overwriting code generated from your transformation model. The TPL compiler converts your TPL code into a code generator.

  You can install your new code generator into OptimalJ with the click of a single button.

- *Javadoc*?OptimalJ provides Javadoc for the code generation and repository APIs. To view this Javadoc, select **View>Documentation Indices>OptimalJ API**.
  - Code generation API?if you prefer to create implementation patterns using Java, a code generation API is provided enabling this.
  - Repository API?the API of the OptimalJ repository is also provided - this Javadoc provides the same information as can be found by using the Explorer [Meta Model].

Step 1 - Create an implementation pattern package structure

All models, including transformation models, are structured inside packages. In this tutorial, a single package is sufficient.

1. Create a new Architecture Edition project:

1. Start the New OptimalJ Project wizard by selecting **Project>New OptimalJ Project**.
2. Name the project `ImplementationPatternTutorial`. Click **Next**.
3. Select **Experiment with One or More Example Models**. Click **Next**.
4. Select **CRM Examples (Sample Domain Model)**, and click **Next**.
5. Select **OptimalJ Metamodel Sources**, and click **Finish**.

Note: In any architecture edition project, the metamodel sources must be mounted.

6. In the Explorer [Transformation Model], select the Transformation node and choose **Popup Menu>New TransformationPackage**.
7. Name the package `webhelper`.
8. Click **Finish**.

Step 2 - Create an implementation pattern module

Implementation pattern modules define the deployment unit of your pattern. When you compile and install your pattern, it is the module (and all of the objects that it contains) that is installed.

1. Select the webhelper package, and choose **Popup Menu>New Child>ImplementationPatternModule**.
2. Name the new module webhelperModule.
3. Click **Finish**.

Step 3 - Create implementation patterns

ImplementationPattern objects define the model features for which you wish to generate code. Each ImplementationPattern object specifies a model feature for which code is to be generated. Your implementation pattern as a whole is defined as a set of ImplementationPattern objects belonging to a module.

1. Select the webhelper package, and choose **Popup Menu>New Child>ImplementationPattern**.
2. Set the following properties:

- Name the new pattern object WebhelperPattern.
- Ensure that the pattern object belongs to the *module* webhelperModule.
- Ensure that the *patternLanguage* is TPL (this is the default).
- Set the *source* to WEB.WEBComponent.

Note: Setting source to WEBComponent indicates that this pattern object generates code for each WEBComponent in the Web model. This is appropriate, as you are generating a utility class for each WEBComponent.

3. Click **Finish**.

Step 4 - Generate code

Your transformation model is complete, and you can now generate code from your model. The generated code includes deployment code to register the pattern in OptimalJ, and TPL and Java files you can use to implement your pattern.

1. Select the webhelper package and choose **Popup Menu>Generate Code**.

Step 5 - Edit the generated TPL

In this step you define a TPL template containing the skeleton code of a Java class, and some TPL expressions to insert values from the Web model.

1. Select `WebhelperPattern` and choose **Popup Menu>Open WebhelperPattern.tpl**. This opens the generated TPL file for `WebhelperPattern`.
2. Change the lines from `[TEMPLATE...` to `[/TEMPLATE]` to specify skeleton code for the utility class. Use the following code:

```
 1 | [TEMPLATE PUBLIC WEBHELPERPATTERN(WEBComponent wEBComponent)]
 2 | Generating
   |   WEBHELPER code for [wEBComponent.name()]
 3 | [FILE("WebhelperPattern")][wEBComponent.name()]Util.java[/FILE]
 4 | [GUARD("WEBHELPERPATTERN",
   |    wEBComponent)]
 5 | package [wEBComponent.refParent().getRefQualifiedName()];
 6 | //add
   |   your import statements here
 7 | class [wEBComponent.name()]Util {
 8 |    void log() {
 9 |       //add logging code here
10 |    }
11 |    String encrypt(String s) {
12 |      //add encryption code here
13 |      return s;
14 |    }
15 | //add you own methods here.
16 | }
17 | [/GUARD]
18 | [/TEMPLATE]
```

   For more information on TPL statements, see *TPL syntax*.
3. Import the `RefObject` class obtained using `wEBComponent.refParent()`, and which is used to calculate the package name of the Java class. A `RefObject` is a generic object that represents repository objects:

```
[IMPORT com.compuware.repository.reflective.RefObject]
```

Note: Learn more about the features of WEBComponent and RefObject in the API Javadoc.

Step 6 - Add free blocks

The implementation pattern now defines the desired utility class. However, the skeleton code lacks free blocks, so users of your implementation pattern lose their code every time they regenerate their code model!

1. To solve this problem, add free blocks around the `//add...` comments.
2. Each free block inside a template must have a different name. Use the prefix `webhelper:` in the free block name to group all your free blocks under one heading in the model-to-code pop-up menu.

   Your TPL code should be similar to the following:

```
[FREE("webhelper:imports")]
//add your import statements here
[/FREE]
```

   This creates the following model-to-code navigation sequence:

   WEBComponent Popup Menu>Edit Free Blocks in Generated Files>webhelper>WEBComponentNameUtil.java>FreeBlockName

Step 7 - Install and test the pattern

Install and test your implementation pattern, by compiling your code and installing the pattern in OptimalJ.

1. Change to the Explorer [Code Model].
2. Select `MountPoint/webhelper/webhelperModuleTpModule.xmimof` and choose **Pop-up Menu>Execute**. This compiles your pattern and installs it in OptimalJ.

   For more information, see *Compiling an implementation pattern* and *Testing an implementation pattern*.

3. Generate code for your Web model: the WebHelper pattern generates the utility classes for each WEBComponent.

Note: When you generate code, you can receive a message that your software factory has changed. You receive this notice when you have generated code, and then change the set of active implementation patterns. This change could affect the code you have previously generated, so OptimalJ notifies you and gives you the opportunity to cancel the code generation action. For more information, see Software factory.

4. You can now navigate from Web components to the utility classes using the navigation sequence:

   WEBComponent Popup Menu>Edit Free Blocks in Generated Files>WebHelper>WEBComponentNameUtil.java>FreeBlockName
5. Add some code to the free blocks, and regenerate code for your Web model. Verify that your free block code is preserved correctly.

In this tutorial you have created a new implementation pattern that creates a simple utility class for Web components. Your code is generated in addition to the other code generated for Web components.

Further reading

However, you have not influenced the *behavior* of the default Web pattern. Your code is never called from the default Web component files, unless these files are modified to call the new code. While the WebHelper pattern is useful, it requires more work to integrate its code into the code generated by the default Web pattern. You can achieve this effect by learning more about *pattern collaboration* and the join points provided by the OptimalJ implementation patterns.

1.26 Creating technology patterns

Technology patterns enable you to map definitions in models, so that changes in one model (the source model) can be applied as updates to the target model. For example, the DBMS technology pattern maps domain class model elements to DBMS model elements. Using the DBMS technology pattern, you can create a DBMS model from the domain class model, and then update the DBMS model in response to changes in the domain class model.

Technology patterns are defined in OptimalJ using the transformation model. Java code is generated from this model, and the code is compiled into an OptimalJ module (plug-in).

The code generated from the transformation model conforms to the incremental copier API. You can use this API to define the transformation rules for your technology pattern. For more information about incremental copiers, see *Incremental copiers*.

In this tutorial you create a technology pattern called ServiceToEJB that maps DomainServices to EJBSessionComponents. The pattern also specifies that the EJBSessionComponents should have the following:

- A BusinessMethod for each DomainServiceOperation defined in the source DomainService.
- A BusinessMethod named `start()` which takes no parameters and does not provide a return value. (This method claims some resources to be used by the component.)
- A BusinessMethod named `stop()` which takes no parameters and does not provide a return value. (This method releases some resources used by the component.)
- A tag named `ServiceToEJB` with the value of `true`. This tag can be used by an implementation pattern to generate code specific to EJBSessionComponents created by the ServiceToEJB technology pattern.
- The package structure of the source model should be reflected by the target model (so sub-packages of the service package should be mapped to sub-packages of the EJB package).

These requirements are known as transformation rules.

The following diagram shows the overall transformation specified by the ServiceToEJB pattern. The numbers in the diagram correspond to the steps in this tutorial which implement that part of the transformation.

Figure 1-128 ServiceToEJB technology pattern.

Note: The pattern also creates an EJBModule and RemoteInterfaces (required by the EJB model checker). These are omitted from the diagram for clarity. The rules to map the EJBModule are covered in step 18 of this tutorial. RemoteInterfaces are mapped in step 12.

Prerequisites

This tutorial is intended for developers and architects starting to use OptimalJ's technology pattern features.

You must have a strong grasp of the OptimalJ application and domain models, and a good understanding of the tools used to develop technology patterns.

You should be familiar with the MDA architecture, and the role played by transformation patterns in this architecture.

Duration

This tutorial takes approximately two hours to complete.

Objectives

At the end of this tutorial you understand the essential concepts and tasks for developing technology patterns, and have enough knowledge to continue learning about technology pattern development independently.

Step 1 - Create a technology pattern package structure

All models, including technology pattern models, are structured inside packages. In this tutorial, a single package is sufficient.

1. Create a new Architecture Edition project:

1. Start the New OptimalJ Project wizard by selecting **Project>New OptimalJ Project**.
2. Name the project `TechnologyPatternTutorial`. Click **Next**.
3. Select **Experiment with One or More Example Models**. Click **Next**.
4. Select **CRM Examples (Sample Domain Model)**, and click **Next**.
5. Select **OptimalJ Metamodel Sources**, and click **Finish**.

Note: In any architecture edition project, the metamodel sources must be mounted.

6. In the Explorer [Transformation Model], select the Transformation node and choose **Popup Menu>New TransformationPackage**.
7. Name the package `servicetoejb`.
8. Click **Finish**.

Step 2 - Create a technology pattern module

Technology pattern modules define the deployment unit of your pattern. The module assembles a group of TechnologyPattern objects into an OptimalJ plug-in. When you compile and install your pattern, it is the TechnologyPatternModule and its associated TechnologyPatterns that is installed.

1. Select the `servicetoejb` package, and choose **Popup Menu>New Child>TechnologyPatternModule**.
2. Name the new module `ServiceToEJBModule`.
3. Click **Finish**.

Step 3 - Create technology patterns

TechnologyPattern objects define a mapping between a source and a target model package, and provide rules to define mappings for the children of the model packages.

1. Select the `servicetoejb` package, and choose **Popup Menu>New Child>TechnologyPattern**.
2. Set the following properties:

- *Name* - `ServiceToEJBPattern`.
- *module* - `ServiceToEJBModule`.

3. Click **Finish**.

Step 4 - Generate code

Your technology pattern model is complete, and you can now generate code from your model. The generated code includes deployment code to register the pattern in OptimalJ, and Java files you can use to implement your pattern.

1. Select the `servicetoejb` package and choose **Popup Menu>Generate Code**.

Step 5 - Edit the generated code

1. Select `ServiceToEJBPattern` and choose **Popup Menu>Open ServiceToEJBPattern.java**. This opens the generated Java file for `ServiceToEJBPattern`.
2. `ServiceToEJBPattern.java` is an incremental copier. For more information on the structure of an incremental copier file, see *Structure of an incremental copier*.
3. Add these import statements to `ServiceToEJBPattern.java`:

```
 1 | //Utility classes
 2 | import com.compuware.alturadev.cm.utils.ContentFilterManager;
 3 | import
   |   com.compuware.alturadev.models.optimalFoundation.ContentFilter;
 4 | import
   |   com.compuware.corbalibrary.types.ClassUList;
 5 | import com.compuware.alturadev.models.optimalFoundation.
   |   collections.AlturaPatternSet;
 6 | import com.compuware.alturadev.models.optimalFoundation.
   |   collections.AlturaMappingBaseUList;
 7 | import com.compuware.alturadev.models.optimalFoundation.
   |   AlturaMapping;
 8 | //Source model elements
 9 | import com.compuware.alturadev.models.domainServices.DomainService;
10 | import
   |   com.compuware.alturadev.models.domainServices.DomainServiceOperation;
11 | import
   |   com.compuware.alturadev.models.domainServices.DomainServiceParameter;
12 | import
   |   com.compuware.alturadev.models.optimalFoundation.DomainElement;
13 | import
   |   com.compuware.alturadev.models.optimalFoundation.AlturaElement;
14 | import
   |   com.compuware.alturadev.models.domainDataTypes.DomainDataType;
15 | import
   |   com.compuware.alturadev.models.domainServices.ServiceContentFilter;
16 | //Target
   |   model elements
17 | import com.compuware.alturadev.models.eJB.EJBModule;
18 | import com.compuware.alturadev.models.eJB.EJBSessionComponent;
19 | import
   |   com.compuware.alturadev.models.eJB.BusinessMethod;
20 | import com.compuware.alturadev.models.eJB.EJBComponentParameter;
21 | import
   |   com.compuware.alturadev.models.eJB.RemoteInterface;
22 | import com.compuware.alturadev.models.optimalFoundation.
   |   ApplicationElement;
```

```
23 | import com.compuware.alturadev.models.optimalFoundation.
   |   Tag;
24 | import com.compuware.alturadev.models.eJBDataTypes.EJBDataType;
25 | import
   |   com.compuware.alturadev.models.eJB.EjbContentFilter;
```

4. In the Explorer [Code Model] mount the following JARs:

- *OptimalJInstallationDirectory*/modules/ojDomainService.jar

These JARs contain the objects referred to by the above `import` statements.

Step 6 - Constrain the source and target packages

The transformation model only allows you to define the source and target types for a TechnologyPattern (in this case, the pattern maps a ModelPackage to a ModelPackage). However, you need to further constrain this, so that the pattern can only be applied to source models that contains domain service model elements, and to target models that contain EJB model elements.

OptimalJ calls the incremental copier method `canCopy()` to determine if an incremental copier can be applied to a given source and target pair.

1. In `ServiceToEJBPattern.java`, select the method `canCopy()` and replace it with the following code:

```
1 |  public boolean canCopy(RefObject source, RefObject target)
  |    {
2 |     return (
3 |          source instanceof ModelPackage
4 |          && ((ModelPackage)source).contentFilter()
  |            instanceof ServiceContentFilter
5 |          && target instanceof ModelPackage
6 |          && ((ModelPackage)target).contentFilter()
  |            instanceof EjbContentFilter);
7 |  }
```

Step 7 - Define rule: ModelPackage maps to ModelPackage

ServiceToEJB maps sub-packages of the service package to sub-packages of the EJB package. Define this rule by adding a `getChildClasses()` method to `ServiceToEJBPattern.java`.

For more information on getChildClasses() rules, see *getChildClasses() rules*.

1. In `ServiceToEJBPattern.java`, insert the following method directly after the `canCopy()` method:

```
 1 | //-------------------------------------------------------------------------
-----
 2 | //--
   |   getChildClasses() rules
 3 | //-- These rules map source model elements to target model
   |   elements
 4 | //-------------------------------------------------------------------------
-----
 5 |
 6 | /**
   |   Declares that a source ModelPackage maps to a target ModelPackage.
   |    */
 7 | public Class getChildClasses(ModelPackage source,
 8 |                              String       referenceName,
   |
 9 |                              ModelPackage sourceParent,
   |
10 |                              ModelPackage targetParent)
   |                                {
11 |     return ModelPackage.class;
12 | }
```

Step 8 - Define rule: DomainService maps to EJBSessionComponent

ServiceToEJB maps each DomainService to an EJBSessionComponent. Define this rule by adding another getChildClasses() rule.

1. In `ServiceToEJBPattern.java`, insert the following method:

```
 1 | /** DomainService maps to a target EJBSessionComponent.
   |    */
 2 | public Class getChildClasses(DomainService source,
 3 |                              String        referenceName,
   |
 4 |                              ModelPackage  sourceParent,
   |
```

```
5 |                              ModelPackage  targetParent)
  |                                {
6 |     return EJBSessionComponent.class;
7 | }
```

Step 9 - Define rule: DomainOperation maps to BusinessMethod

ServiceToEJB maps each DomainServiceOperation to a BusinessMethod. Define this rule by adding another getChildClasses() rule.

1. In `ServiceToEJBPattern.java`, insert the following method:

```
1 | /** DomainServiceOperation maps to a target BusinessMethod.
  |    */
2 | public Class getChildClasses(DomainServiceOperation source,
  |
3 |                              String                 referenceName,
  |
4 |                              DomainService          sourceParent,
  |
5 |                              EJBSessionComponent    targetParent)
  |                                {
6 |     return BusinessMethod.class;
7 | }
```

Step 10 - Define rule: DomainServiceParameter maps to EJBComponentParameter

Map the parameters of the DomainServiceOperation to EJBComponentParameters.

1. In `ServiceToEJBPattern.java`, insert the following method:

```
1 | /** DomainServiceParameter maps to a target EJBComponentParameter.
  |    */
2 | public Class getChildClasses(DomainServiceParameter source,
  |
3 |                              String                 referenceName,
  |
4 |                              DomainServiceOperation sourceParent,
  |
5 |                              BusinessMethod         targetParent)
```

```
   |                                {
 6 |     return EJBComponentParameter.class;
 7 | }
```

Step 11 - Define rule: DomainService maps to two BusinessMethods, a Tag, and a RemoteInterface

For each EJBSessionComponent mapped by ServiceToEJB, there should be `start()` and `stop()` BusinessMethods, and a Tag named `ServiceToEJB`. A RemoteInterface should also be provided for the EJBSessionComponent.

This step uses the Empty type as a source object to indicate that the target types should be mapped whenever a DomainService is mapped to an EJBSessionComponent.

The rule calls a `superDeepCopyChild()` method to indicate that the DomainService maps to children of the EJBSessionComponent. For more information about establishing mappings manually, see *Mapping target objects manually*.

1. In `ServiceToEJBPattern.java`, insert the following method:

```
 1 | /** DomainService maps to children of EJBSessionComponent.
   |    */
 2 | public Class getChildClasses(Empty              source,
   |
 3 |                              String             referenceName,
   |
 4 |                              DomainService      sourceParent,
   |
 5 |                              EJBSessionComponent targetParent)
   |                                {
 6 |     //superDeepCopyChild() copies a source element to
 7 |     //children of the targetParent.
 8 |     //In this case we are copying the sourceParent element
 9 |     //to
   |       children of targetParent.
10 |     superDeepCopyChild(sourceParent, "feature", targetParent);
11 |     superDeepCopyChild(sourceParent,
   |        "taggedValue", targetParent);
12 |     superDeepCopyChild(sourceParent, "manifestation", targetParent);
```

```
13 |     return
   |       null;
14 | }
```

Step 12 - Define rule: EJBSessionComponent has a BusinessMethod named start(), a Tag named ServiceToEJB, and a RemoteInterface

For each EJBSessionComponent mapped by ServiceToEJB, there should be a `start()` BusinessMethod and a Tag named ServiceToEJB.

This step defines a rule that is called as a result of step 11, which maps DomainService to children of EJBSessionComponent. (If the rule defined in step 11 is deleted, the rule below would never be executed).

1. In `ServiceToEJBPattern.java`, insert the following method:

```
 1 | /** DomainService maps to: BusinessMethod, and Tag. */
 2 |      public
   |        Class getChildClasses(DomainService       source,
 3 |                                  String              referenceName,
   |
 4 |                                  ModelPackage        sourceParent,
   |
 5 |                                  EJBSessionComponent targetParent)
   |                                    {
 6 |     Class target = null;
 7 |     //if rule is called for taggedValue collection... (add
   |       Tag)
 8 |     if( referenceName.equals("taggedValue") ) {
 9 |         target = Tag.class;
10 |     } else if (referenceName.equals("manifestation")) {
11 |     //.
   |       ..else if rule is called for manifestation collection.
   |       ..
12 |         target = RemoteInterface.class;
13 |     } else if ( referenceName.equals("feature") ) {
14 |     //...else rule is called for feature collection
15 |         //Add BusinessMethod
16 |         target = BusinessMethod.class;
17 |     }
18 |     return target;
```

```
19 | }
```

2. Setting the names of the target objects is handled in later steps.

Step 13 - Define rule: EJBSessionComponent has a BusinessMethod named stop()

This step looks similar to step 12, but requires a completely different implementation! This is because getChildClasses() rules cannot map the same source to two or more targets of the same type. For more information on getChildClasses() rules, see *getChildClasses() rules*.

To map a second instance of BusinessMethod, you must use the context stack to distinguish the second instance, `stop()`, from the first instance, `start()`. For more information on the context stack, see *Context stack*.

Note: The code in this step modifies the rule defined in step 12.

1. In `ServiceToEJBPattern.java`, update the method created in step 12:

```
 1 | /** DomainService maps to: BusinessMethod, BusinessMethod,
   |    and Tag. */
 2 |      public Class getChildClasses(DomainService       source,
   |
 3 |                                   String             referenceName,
   |
 4 |                                   ModelPackage       sourceParent,
   |
 5 |                                   EJBSessionComponent targetParent)
   |                                     {
 6 |     Class target = null;
 7 |     //START NEW CODE
 8 |     //if contextStack empty, repeat this rule to create
   |       a 2nd BusinessMethod
 9 |     if( getSourceContext().isEmpty() ) {
10 |         superDeepCopyChildWithContext(source, "feature",
   |
11 |             targetParent, sourceParent.namespace());
12 |     }
13 |     //END NEW CODE
```

```
14 |     //if rule is called for taggedValue collection... (add
   |       Tag)
15 |     if( referenceName.equals("taggedValue") ) {
16 |         target = Tag.class;
17 |     } else if (referenceName.equals("manifestation")) {
18 |     //.
   |       ..else if rule is called for manifestation collection.
   |       ..
19 |         target = RemoteInterface.class;
20 |     } else if ( referenceName.equals("feature") ) {
21 |     //...else rule is called for feature collection
22 |         //Add two BusinessMethods
23 |         target = BusinessMethod.class;
24 |     }
25 |     return target;
26 | }
```

The rule now calls a `superDeepCopyChildWithContext()` method to reapply the rule with an additional object in the context stack. The call to `superDeepCopyChildWithContext()` is contained by an `if` statement to prevent an infinite loop.

2. Setting the name of the BusinessMethod to `stop()` is handled in another rule (see below).
3. This rule completes the first phase of the transformation - namely, all rules that establish mappings from instances in the source model to instances in the target model are complete. (Creating a default EJBModule is an exception - because this is the most complex part of the technology pattern, this task is covered in step 18 instead).

   In the following steps you provide rules to set the properties of the target model elements mapped by the rules defined in steps 7 to 13.

Step 14 - Define rule: DomainDataType maps to EJBDataType

This step declares that all references to DomainDataTypes should be mapped to EJBDataTypes using a getReferenceClass() rule. You then call another pattern to map the instances of these data types (for instance, mapping the domain version of the String data type to the EJB version of this data type).

1. In ServiceToEJBPattern.java, add the following method directly after the getChildClasses() rules:

```
 1 | //-------------------------------------------------------------------
-----
 2 | //--
   |   getReferenceClass() rules
 3 | //-- These rules handle non-composite references.
 4 | //-------------------------------------------------------------------
-----
 5 |
 6 | /**
   |   DomainDataType maps to a target EJBDataType. */
 7 | public Class getReferenceClass(DomainDataType     source,
   |
 8 |                                String             referenceName,
   |
 9 |                                DomainElement      sourceReferrer,
   |
10 |                                ApplicationElement targetReferrer)
   |                                  {
11 |     return EJBDataType.class;
12 | }
```

2. This getReferenceClass() rule establishes a mapping from DomainDataType to EJBDataType; the mappings for instances of these data types is established in the OptimalJ meta models.

   To use these predefined mappings of data types, add the following method to ServiceToEJBPattern.java:

```
protected AlturaPatternSet getDependentForwardPatterns()
  {
    AlturaPatternSet resultSet = super.getDependentForwardPatterns();
    resultSet.
      add(getMappingPattern("domainToEjbMapping"));
    return resultSet;
```

```
}
```

Note: To improve the readability of your code, you can add this method to the `Pattern dependencies` section of the Java file, which is set aside for methods dealing with relationships with other patterns. To find this section, search your file for the string `Pattern dependencies`.

Step 15 - Define rule: Set BusinessMethod.name

In steps 12 and 13 you mapped two BusinessMethods, `start()` and `stop()`. However, the names of these methods are not defined by the mapping rules. You need to define a further set of rules to set the properties of the mapped elements. These rules are called copyStructuralFeatures() rules.

1. In `ServiceToEJBPattern.java`, add the following method directly after the getReferenceClass() rule:

```
 1 | //-------------------------------------------------------------------------
-----
 2 | //--
   |   copyStructuralFeatures() rules
 3 | //-- These rules set property values on mapped target model
   |   elements.
 4 | //-------------------------------------------------------------------------
-----
 5 |
 6 | /**
   |   Set the name of BusinessMethods mapped from DomainService.
   |   */
 7 | public void copyStructuralFeatures(DomainService  source,
 8 |                                    BusinessMethod target)
   |                                      {
 9 |     String newName = "start";
10 |     AlturaMappingBaseUList ambul = target.getTargetMapping(pattern,
   |        false);
11 |     //We know that target is only mapped by one pattern.
   |       ..
12 |     AlturaMapping am = (AlturaMapping)(ambul.iterator().
   |       next());
```

```
13 |     java.util.Collection c = am.getSourceContext();
14 |     if( !c.isEmpty() ) {
15 |         newName="stop";
16 |     }
17 |     target.setName(newName);
18 | }
```

The rule only applies to BusinessMethods mapped from a DomainService source - so it only affects the BusinessMethods mapped as a result of steps 12 and 13. The rule sets the name to `stop` if the if target is mapped to more than one source item (which is the case when `target` is mapped using `superDeepCopyWithContext()`), otherwise the name is set to `start`.

Step 16 - Define rule: Set Tag.name and Tag.value

In step 12 you mapped a Tag named `ServiceToEJB`. However, the name of the Tag is not defined by the mapping rule. In this step you define a copyStructuralFeatures() rule to set the Tag name and the tag value.

1. In `ServiceToEJBPattern.java`, add the following method directly after the getReferenceClass() rule:

```
 1 | /** Set the name and value of Tags mapped from Empty. */
 2 | public
   |   void copyStructuralFeatures(DomainService  source,
 3 |                                    Tag           target)
   |                                      {
 4 |     target.setName("ServiceToEJB");
 5 |     target.setValue("true");
 6 | }
```

The rule only applies to Tags mapped from a DomainService source - so it only affects the Tags mapped as a result of step 12.

Step 17 - Define rule: Set ModelPackage.contentFilter

ServiceToEJB maps ModelPackage to ModelPackage (see step 7). You must define another copyStructuralFeatures() rule to set the content filter of the target model package to EJB.

1. In `ServiceToEJBPattern.java`, add the following method directly after the getReferenceClass() rule:

```
/** Set ModelPackage.contentFilter = EjbContentFilter.
```

```
  */
public void copyStructuralFeatures(ModelPackage source,
                                   ModelPackage target)
                                     {
    target.setContentFilter(ContentFilterManager.getInstance().
      get(EjbContentFilter.class));
}
```

Step 18 - Define rule: Create a default EJBModule

A valid EJB model requires that each EJB component should be attached to an EJBModule. Generating this EJBModule is the most complex part of this transformation, because:

- the EJBModule should only be generated if EJBSessionComponents are generated
- Only one default EJBModule should be generated - regardless of the structure of the source model
- After generating the EJBModule, each EJBSessionComponent should be attached to the module (if the component is not already a member of another module)

1. To ensure that the default EJBModule is only generated when the technology pattern has generated EJBSessionComponents, modify the rule created in step 11, because this rule is applied after an EJBSessionComponent is mapped:

```
1 | //START NEW CODE #1
2 | private boolean defaultModuleRuleAppliedBefore = false;
3 | //END
  |   NEW CODE #1
4 | /** DomainService maps to children of EJBSessionComponent.
  |
5 |  When at least one EJBSessionComponent is mapped, there
  |    should be a
6 |  default EJBModule.*/
7 | public Class getChildClasses(Empty              source,
  |
8 |                              String             referenceName,
  |
9 |                              DomainService      sourceParent,
```

```
   |
10 |                                EJBSessionComponent targetParent)
   |                                  {
11 |     superDeepCopyChild(sourceParent, "feature", targetParent);
12 |     superDeepCopyChildWithContext(sourceParent,
   |        "taggedValue",
13 |         targetParent, sourceParent.refParent() );
14 |     superDeepCopyChild(sourceParent, "manifestation", targetParent);
15 | //START
   |   NEW CODE #2
16 |     //only apply this mapping once - additional calls are
   |       redundant
17 |     if(defaultModuleRuleAppliedBefore == false) {
18 |         defaultModuleRuleAppliedBefore = true;
19 |         superDeepCopyChild(getRootSource(), "ownedElement",
   |
20 |             pattern.getTarget());
21 |     }
22 | //END NEW CODE #2
23 |     return null;
24 | }
```

The code uses the `pattern` variable, which is available in all incremental copiers, and is a reference to the pattern node which represents the technology pattern in the target model. The pattern node contains references to the topmost source and target objects. For more information, see *Pattern nodes*.

The code adds a call to `superDeepCopyChild()` to map the topmost package in the source model to children of the topmost package in the target model.

This `superDeepCopyChild()` call would be made every time an EJBSessionComponent is mapped, with the result that the same set of mappings would be made repeatedly. This is not a problem as the incremental copier disregards redundant mappings. However, to prevent unnecessary processing, the call is hidden inside an `if` block to ensure that the call is only made once.

2. To map a default EJBModule, modify the rule for mapping ModelPackages (defined in step 7):

```
 1 | /** ModelPackage maps to a target ModelPackage or EJBModule.
   |    */
```

```
 2 | public Class getChildClasses(ModelPackage source,
 3 |                              String       referenceName,
   |
 4 |                              ModelPackage sourceParent,
   |
 5 |                              ModelPackage targetParent)
   |                                {
 6 | //START NEW CODE
 7 |     Class target = null;
 8 |     if(source == getRootSource()) {
 9 |         target = EJBModule.class;
10 |     } else {
11 |         target = ModelPackage.class;
12 |     }
13 |     return target;
14 | //END NEW CODE
15 | }
```

The new code returns an EJBModule when the `source` is the topmost source model package.

Under default circumstances, this rule is not called for the topmost source model package - transformation rules are only applied to the *children* of this package. However, the call to `superDeepCopyChild()` applies this rule to the topmost source package, and maps it to an EJBModule attached as a child to the topmost target model package.

3. To add EJBSessionComponents to the EJBModule, keep a reference to the EJBModule (alternatively it would be possible, but much more complicated, to obtain this module by navigating the mapping objects). The easiest way to get a reference to the EJBModule is to define a new rule:

```
1 | private EJBModule defaultEJBModule = null;
2 | /** Use an empty rule to store a reference to the default
3 |  EJBModule for later use. */
4 | public Class getChildClasses(Empty        source,
5 |                              String       referenceName,
  |
6 |                              ModelPackage sourceParent,
  |
```

```
 7 |                                EJBModule    targetParent)
   |                                  {
 8 |     defaultEJBModule = targetParent;
 9 |     return null;
10 | }
```

Rules with an Empty source are called directly after `sourceParent` is mapped to `targetParent`. In this case, `targetParent` is a reference to the default EJBModule, and this reference is stored in instance variable `defaultEJBModule` for later use.

4. To add the EJBSessionComponents to the default EJBModule, define a new copyStructuralFeatures() rule to set EJBSessionComponent.module property:

```
1 | /** Set EJBSessionComponent.module = defaultEJBModule.
  |   */
2 | public void copyStructuralFeatures(DomainService
  |   source,
3 |                                    EJBSessionComponent
  |                                      target) {
4 |     if(target.getModule() == null) {
5 |         target.setModule(defaultEJBModule);
6 |     }
7 | }
```

You must test for whether EJBSessionComponent.module is `null`, because the component could be attached to another, non-default module!

Step 19 - Declare dependencies on other OptimalJ modules

ServiceToEJB uses model elements defined in the EJB and Service models, as well as generic model elements such as Tag, ApplicationElement, and DomainElement. This means that ServiceToEJB depends on the OptimalJ modules that define these elements. (This means that the pattern would not work in a version of OptimalJ that does not provide a service model!)

You declare such dependencies in the pattern module's manifest file.

1. In the Explorer [Transformation Model], right-click ServiceToEJBModule, and choose **Edit Generated Files>ServiceToEJBModule.mf** from the pop-up menu.

2. In `ServiceToEJBModule.mf`, edit the `OpenIDE-Module-Module-Dependencies` line:

```
1 | OpenIDE-Module-Module-Dependencies: org.openide.debugger
  |   > 1.0,
2 |                                    org.openide.compiler
  |                                      > 1.0,
3 |                                    org.openide.execution
  |                                      > 1.0,
4 |                                    org.openide.io >
  |                                      1.0,
5 |                                    com.compuware.alturainfra.
  |                                      module,
6 |                                    com.compuware.netbeans.
  |                                      modules.mofToolBase,
7 |                                    com.compuware.netbeans.
  |                                      modules.mofToolGenerator,
8 |                                    com.compuware.alturadev.
  |                                      cm.netbeans,
9 |                                    com.compuware.alturadev.
  |                                      service,
10 |                                   com.compuware.alturadev.
  |                                      ejb.netbeans,
11 |                                   com.compuware.alturadev.
  |                                      domain.netbeans,
```

Step 20 - Compile and install ServiceToEJB

Compile and install your technology pattern by compiling your code and installing the pattern in OptimalJ.

1. Change to the Explorer [Code Model].
2. Select `MountPoint/servicetoejb/ServiceToEJBModuleTpModule` and choose **Pop-up Menu>Execute**. This compiles your pattern and installs it in OptimalJ.

   For more information, see *Compiling a transformation pattern* and *Testing a transformation pattern*.

Step 21 - Test: Define a domain service model

To test your technology pattern you need a domain service model. Either use an existing domain service model, or create a suitable test model.

Your test model should define at least one DomainService. In addition, the model should contain a DomainServiceOperation which accepts at least one DomainServiceParameter.

Step 22 - Test: Map the domain service package to the EJB package

To test your technology pattern, you must apply the pattern to an EJB model package.

1. In the Explorer [Application Model], right-click the application ModelPackage, and choose **New Child>ModelPackage** from the pop-up menu. This starts the Create a Model Package wizard.
2. Set the following properties:

- **Name** - `servicetoejbtest`
- **Content Filter** - `ejb`

3. Click **Next**. This opens the Select source ModelPackages and patterns wizard step.
4. Click **Add** to open the Select Source ModelPackage window.
5. Select the service package and click OK.
6. Select `ServiceToEJBPatternPattern` from the drop-down list.
7. Click **Finish**.

Step 23 - Test: Apply the technology pattern

1. In the Explorer [Application Model], right-click the node application.servicetoejbtest.ServiceToEJBPatternPattern, and choose **Update Model** from the pop-up menu.
2. Inspect the resulting EJB model.

   Ensure that an EJBSessionComponent has been created for each DomainService, and that the methods and tags specified by the pattern are present on the components.
3. Verify the structure of your model.

   In the Explorer [Application Model], right-click the node application.servicetoejbtest, and choose **Check Model** from the pop-up menu.
4. Re-apply the technology pattern.

A technology pattern should have a reproducible effect. If you regenerate a target model, it should only change when the source model has changed, and should *not* change because the technology pattern has been re-applied. Test this by regenerating your EJB model: the EJB model should remain unchanged.

In this tutorial you have created a new technology pattern that creates a simple EJB model containing several EJBSessionComponents with some predefined methods and tags.

Further reading

Consider the following questions:

- How could you ensure that only certain DomainServices are mapped to the EJB model? For example, you might prefer that ServiceToEJB should ignore any DomainService that uses a DomainView (because ServiceToEJB does not handle DomainViews).
- How can you prevent naming clashes? The domain service model might define methods named start() or stop(). This would clash with the names of the methods generated by ServiceToEJB.
- How could you add support for DomainViews to this technology pattern?

For more information, see also *Extending a technology pattern* under technology patterns.

1.27 Creating metamodels

You can add new model type to OptimalJ by creating a metamodel that describes the new model type. A metamodel is a class model where each class describes a model element type. A model conforms to the metamodel if it contains only model elements that conform to the model element types (and their constraints) defined in the metamodel.

Metamodels are defined using MOF, a modeling language developed by the OMG for metamodeling. For more information on metamodels, see *Metamodeling*.

Prerequisites

To complete this tutorial, you should understand:

- MDA and how OptimalJ implements MDA.
- The default model types provided with OptimalJ, such as the EJB and Web models.
- How to define UML or MOF class models.
- Java.
- The repository API.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial you define a metamodel called the RangeModel, which is a model for defining ranges of integer values. The range model has one model element type, Range, which has three attributes, minimum, maximum, and name. Range elements can restrict another Range, where the Range is a subset of the restricting Range.

For example, you could model a marking system as a set of ranges:

- PercentageRange: 0 to 100
- Fail: 0 to 49
- Pass: 50 to 69
- Second-class pass: 70 to 79
- First-class pass: 80 to 100
- In this example, PercentageRange restricts all the other ranges, so each pass or fail category must be a range of valid percentages.
- You can model this marking system as a test for your new metamodel.

There are two constraints that you will implement using a model checker:

- Range.minimum must be less than Range.maximum for every instance of Range.
- The minimum and maximum values for a Range must fall within the bounds of any restricting Range.

Step 1 - Create a RangeModel metamodel

In this step you create a project and a `RangeModel` metamodel.

1. Create a new Architecture Edition project named `MetaModelingTutorial`.

In the New OptimalJ Project wizard, mount the CM Metamodel sources.

Note: In any architecture edition project, the metamodel sources must be mounted.

2. In the Explorer [Meta Model], right-click the Meta node. From the pop-up menu, choose **New Package**. Name the new Package `RangeModel`.

   Click **Finish**.
3. The wizard creates a new metamodel for you.

   Expand the RangeModel Package and view the default contents of the model. For more information on these elements, see *Defining a metamodel*.
4. Select the Tag org.omg.mof.idl_prefix. Set the *values* property to `com.compuware.alturasample.models`. This sets the location of the code generated from this metamodel.

Step 2 - Create a metaclass named Range

In this step you create the `Range` metaclass and its attributes.

For a more detailed description of creating metaclasses, see *Creating metaclasses*.

1. Right-click the Package `RangeModel`.
2. From the pop-up menu, choose **New Child>Class**. Name the new Class `Range`.
3. Right-click `Range`. From the pop-up menu, choose **New Child>Attribute**. Create an Attribute named `minimum`.
4. Right-click `Range`. From the pop-up menu, choose **New Child>Attribute**. Create an Attribute named `maximum`.
5. Select the Attribute named `minimum`. Set the *type* property to `Foundation.Data_Types.Integer`.

   You can also browse to this data type by selecting the *type* property and clicking **...**. This opens the Type selection dialog box.
6. Select the Attribute named `maximum`. Set the *type* property to `Foundation.Data_Types.Integer`.
7. Right-click the `RangeModel` Package. From the pop-up menu, choose **Check Model**.

   The model checker should report `Model is OK` in the Output Window [ModelChecker].

Step 3 - Create an Association named RestrictedRange

In this step you create the RestrictedRange Association.

For a more detailed description of creating Associations, see *Adding associations between metaclasses*.

1. Right-click the Package RangeModel.
2. From the pop-up menu, choose **New Child>Association**. Name the new Association RestrictedRange.
3. Right-click RestrictedRange. From the pop-up menu, choose **New Child>AssociationEnd**. Create an AssociationEnd named restrictedBy.

- Set the type of the restrictedBy to RangeModel.Range.
- Set the *multiplicity* of restrictedBy to zero or one.

4. Right-click RestrictedRange. From the pop-up menu, choose **New Child>AssociationEnd**. Create an AssociationEnd named subRanges.

- Set the type of subRanges to RangeModel.Range.
- Set the *multiplicity* of subRanges to zero or more, unordered, nonunique.

5. Examine the Class Range. Reference nodes corresponding to the AssociationEnds have been added to Range.
6. Right-click the RangeModel Package. From the pop-up menu, choose **Check Model**.

   The model checker should report Model is OK in the Output Window [ModelChecker].

Step 4 - Define supertypes for the Range metaclass

In this step you define the supertypes for the Range metaclass. The supertypes added in this step provide a name attribute, and enable OptimalJ to display the instances of Range in the Explorer [Application Model].

The superclasses used in this step are part of the OptimalFoundation metamodel. For more information, see *OptimalFoundation*.

1. Select the Class Range.
2. Set the supertypes of the Range to OptimalFoundation.AlturaElement, OptimalFoundation.AlturaTopElement.
3. Right-click the RangeModel Package. From the pop-up menu, choose **Check Model**.

The model checker should report `Model is OK` in the Output Window [ModelChecker].

Step 5 - Add a constraint: MinimumLessThanMaximum

In this step you define the constraint that the minimum value for a `Range` must be less than (or equal to) the maximum value.

For more information on defining class constraints, see *Constraining Classes*.

1. Right-click the Class `Range`.
2. From the pop-up menu, choose **New Child>Constraint**.
3. Create a Constraint named `MinimumLessThanMaximum`.

- Set the *constrainedElements* property to `RangeModel.Range`.
- Set the *language* property to `Java`.
- Set the *expression* property to:

```
return (element.getMinimum() >
element.getMaximum());
```

Note: The expression must be a boolean expression returning `true` when there is an error condition. You can refer to instances of the constrained element using the variable `element`.

4. Right-click the `RangeModel` Package. From the pop-up menu, select **Check Model**.

   The model checker should report `Model is OK` in the Output Window [ModelChecker].

Step 6 - Add a constraint: RangeIsRestricted

In this step you define a constraint that the bounds of a Range must lie within the bounds of a restricting Range.

For more information on defining class constraints, see *Constraining Classes*.

1. Right-click the Class `Range`.
2. From the pop-up menu, choose **New Child>Constraint**.
3. Create a Constraint named `RangeIsRestricted`.

   Set the *constrainedElements* property to `RangeModel.Range`.

   Set the *language* property to `Java`.

   Set the *expression* property to the following:

```
1 | Range restrictingRange = element.getRestrictedBy();
```

```
 2 | boolean errorCondition = false;
 3 | if (restrictingRange != null) {
 4 |    //element bounds must lie within restrictingRange bounds.
 5 |    if( element.getMinimum() < restrictingRange.getMinimum()
   |      ) {
 6 |       errorCondition = true;
 7 |    }
 8 |    if( element.getMaximum() > restrictingRange.getMaximum()
   |      ) {
 9 |       errorCondition = true;
10 |    }
11 | }
12 | return errorCondition;
```

Note: You can refer to instances of the constrained element using the variable `element`*. The expression must be a boolean expression returning* `true` *when there is an error condition.*

4. Right-click the `RangeModel` Package. From the pop-up menu, choose **Check Model**.

   The model checker should report `Model is OK` in the Output Window [ModelChecker].
5. Right-click the `RangeModel` Package. From the pop-up menu, choose **Generate Code**.
6. After code generation is complete, right-click `RangeModel`. From the pop-up menu, choose **Edit Generated Files>RangeModelModelChecker.java**.

   Examine the code generated for your class constraints.

Step 7 - Deploy and test the model type

Deploy and test your new model type, including testing the class constraints by executing the model checker.

Test your metamodel by creating a range model called `markingsystem`:

1. Install the `RangeModel` model type in OptimalJ, so that you can create instances of this model type in the Explorer [Application Model]. For instructions on installing a metamodel, see *Deploying a metamodel*.

2. In the Explorer [Application Model], create a new root ModelPackage named `test`. Use the Three Tier Application Structure.
3. Select the `application` ModelPackage. From the popup menu, choose **New Child>ModelPackage**.

- Set *name* to `markingsystem`.
- Set *contentFilter* to `range_model`.

4. Create the following Range elements:

- PercentageRange: 0 to 100
- Fail: 0 to 49 (restricted by PercentageRange)
- Pass: 50 to 69 (restricted by PercentageRange)
- Second-class pass: 70 to 79 (restricted by PercentageRange)
- First-class pass: 80 to 100 (restricted by PercentageRange)

5. The model specified above is valid, so you can test your model checker.

   Select the `markingsystem` model package. From the pop-up menu, choose **Check Model**.

   The model checker should report `Model is OK` in the Output Window [ModelChecker].
6. Test your model checker by violating the `MinimumLessThanMaximum` constraint. Set the minimum value for Fail to 65.

   Select the `markingsystem` model package. From the pop-up menu, choose **Check Model**.

   The model checker should report an error in the Output Window [ModelChecker].
7. Test your model checker by violating the `RangeIsRestricted` constraint. Set the minimum value for Fail to -10.

   Select the `markingsystem` model package. From the pop-up menu, choose **Check Model**.

   The model checker should report an error in the Output Window [ModelChecker].

In this tutorial you created a new metamodel, installed this model type in OptimalJ, and created a model conforming to the new model type. While the metamodel is relatively simple, it demonstrates how to define new metaclasses, attributes, associations, and model checkers.

Further reading

For more information, see also *Metamodeling* and *Transformation patterns*.

- *Managing metamodel versions*
- *Behavioral features in metamodels*
- *Class constraints*
- *StructuralFeature constraints*
- *Persistence of associations*
- *Reference specialization*
- *Transformation patterns and metamodels*

1.28 Changing the default OptimalJ metamodels

This tutorial shows you how to change the default model types of OptimalJ by editing the metamodels that describe the model type.

For more information on metamodels, see *Metamodeling*.

Prerequisites

To complete this tutorial, you should understand:

- MDA and how OptimalJ implements MDA.
- The default model types provided with OptimalJ, such as the EJB and Web models.
- How to define UML or MOF class models.
- Java.
- The repository API.

Note: This tutorial does not cover metamodeling in detail, as it focuses on the procedure for installing updated versions of the default metamodels. For a general tutorial on metamodeling, see Creating metamodels.

Duration

This tutorial takes approximately one hour to complete.

Objectives

In this tutorial you update the EJB metamodel, adding the following features to the model:

- Add a boolean attribute `log` to the Class `EJBComponent`.

Step 1 - Create a new project

In this step you create a new Architecture Edition project.

1. Create a new Architecture Edition project:

1. Start the New OptimalJ Project wizard by choosing **Project>New OptimalJ Project**.
2. Name the project `EJBMetaModelingTutorial`. Click **Next**.
3. Select **Experiment with One or More Example Models**. Click **Next**.
4. Select **CRM Examples (Sample Domain Model)**, and click **Next**.
5. Select **OptimalJ Metamodel Sources**, and click **Finish**.

Note: In any architecture edition project, the metamodel sources must be mounted.

Step 2 - Update the EJB metamodel

In this step you add a new attribute to the EJBComponent metaclass, which is part of the EJB metamodel.

For a more detailed description of creating metaclasses, see *Creating metaclasses*.

1. In the Explorer [Meta Model], expand the Package `EJB`.
2. Search for the metaclass `EJBComponent`.

*Tip: `EJBComponent` is one of the first items in the EJB model, if you have not modified the sorting properties for OptimalJ. If you cannot find `EJBComponent`, right-click on the EJB package, and select **Sort Children>By Name** from the pop-up menu.*

3. Right-click `EJBComponent`. From the pop-up menu, choose **New Child>Attribute**. Create an Attribute named `log`.

4. Select the Attribute named `log`. Set the *type* property to `Foundation.Data_Types.Boolean`.

   You can also browse to this data type by selecting the *type* property and clicking **...**. This opens the Type selection dialog box.
5. Right-click the `EJB` Package. From the pop-up menu, select **Check Model**.

   The model checker should report `Model is OK` in the Output Window [ModelChecker].

Step 3 - Create a patch JAR file

The update made in step 2 to the metamodel must be applied to OptimalJ to have any effect. The modified class files are applied to OptimalJ using OptimalJ's patching mechanism.

In this step you create a patch JAR file.

1. Right-click the root Package `EJB`.
2. From the pop-up menu, choose **Generate Code**.
3. In the Explorer [Code Model], expand the directory `OptimalJUserDirectory/metamodel/com/compuware/alturadev/models`. This contains the files generated from the EJB metamodel.
4. Explore the directory `com.compuware.alturadev.models.eJB` to view the code generated from the EJBComponent metaclass.
5. Right-click the directory `OptimalJUserDirectory/metamodel/com`. From the pop-up menu, choose **New>All Templates**. This opens the New Wizard.
6. In the New Wizard, select the template JAR Archives/JAR Recipe. Click **Next**.
7. In the Basic Recipe Properties window, set **Recipe Name** to `ejbPatch`.

   Note the **Generated JAR Location** setting. This is where the patch JAR file is generated.

   Click **Next**. This opens the Specify JAR Recipe Contents window.
8. In the Specify JAR Recipe Contents window, specify the following contents for the JAR recipe:

- `OptimalJUserDirectory/metamodel/com/compuware/alturadev/models/ImpleJB`

- `OptimalJUserDirectory/metamodel/com/compuware/alturadev/models/eJB/collections`
- `OptimalJUserDirectory/metamodel/com/compuware/alturadev/models/eJB/*.java`

Click **Next**.

9. Click **Finish**.
10. Compile your code and build the patch JAR file:

1. Compiling the EJB metamodel can take over an hour on some systems.

   To improve the performance of the Java compiler, allocate more memory to it. For more information, see *Code generation and customization*.
2. Right-click `ejbPatch.jarContent` and choose **Compile** from the pop-up menu.

Step 7 - Deploy the patch JAR file

In this step you add the patch JAR file to OptimalJ.

1. Exit OptimalJ.
2. In your operating system, go to the directory containing the patch JAR file, `ejbPatch.jar`. Copy this file to the clipboard.
3. Go to the following directory:

   `OptimalJInstallationDirectory/modules/patches/`
4. Create a subfolder named:

   `OptimalJInstallationDirectory/modules/patches/com-compuware-alturadev-ejb-netbeans`

   This subfolder is named after the module identifier of the EJB Component Module (`ejbComponentModule.jar`). OptimalJ loads classes from this location in preference to the classes contained in `ejbComponentModule.jar`.

Note: The next step affects all EJB models loaded in your current OptimalJ project. Do not proceed if your project contains EJB models, because the next step changes the structure of the XCM files used to store EJB model data.

5. Paste the patch JAR file into this folder.
6. Restart OptimalJ.

7. Test your work by creating a small EJB model: select **Model>Update All Models** to generate an EJB model for the CRM sample.
8. Inspect an EJBEntityComponent. The component has a new property *log*, which accepts the values `true` and `false`.

In this tutorial you have altered the default EJB model type by adding a new attribute to all types of EJB component, such as EJBEntityComponents and EJBSessionComponents.

Further reading

The alterations to the EJB metamodel enable you to edit models that use the new attribute. However, the transformation patterns provided by OptimalJ do not use this attribute, so you must write new patterns or extend the existing patterns to take advantage of the changes to the metamodel. For more information, see *Technology patterns* and *Implementation patterns*.

When you change a metamodel, you must ensure that your changes do not break existing OptimalJ code. For example, the technology patterns provided by OptimalJ set the values of attributes in EJB models. If you remove an attribute from the metamodel, then the set method called by the technology pattern is not available in the updated API. For more information on the guidelines you should follow, see *Guidelines for changing the default metamodels*.

For more information on the elements and properties provided by the default OptimalJ metamodels, see *Reference*.

1.29 Installing a local CVS server

To run the CVS tutorials in OptimalJ, you need to interact with a running CVS server. If your CVS administrator has created a test repository, you can use this repository to run the CVS tutorials. As an alternative, you can install a freeware CVS server, CVSNT, on their systems and set up a local repository. This tutorial describes how to install CVSNT on a Windows system. Linux users can install the UNIX/Linux version CVSNT if they wish.

The CVSNT home page is located at *http://www.cvsnt.org/wiki/*.

Prerequisites

- This tutorial assumes that the reader is comfortable with installing and configuring software packages on their system.
- Windows NT, 2000, and XP users will need administrator rights on their local machine in order to complete this tutorial.

Duration

This tutorial takes approximately 30 minutes to complete.

Objectives

In this tutorial, you learn how to:

- Download and install CVSNT.
- Create a CVSNT repository.
- Add users for your CVSNT repository.
- Configure your repository for use with OptimalJ projects.

Step 1 - Download and install CVSNT

Download the CVSNT installer and execute it on your local machine. The CVSNT home page is located at *http://www.cvsnt.org/wiki/*. The detailed steps include a few important hints for running the installer.

1. Using a web browser, browse to *http://www.cvsnt.org/wiki/* and download CVSNT version 2.1.1.
2. Run the `cvsnt-2.1.1.exe` you just downloaded and follow the steps in the CVSNT setup wizard.

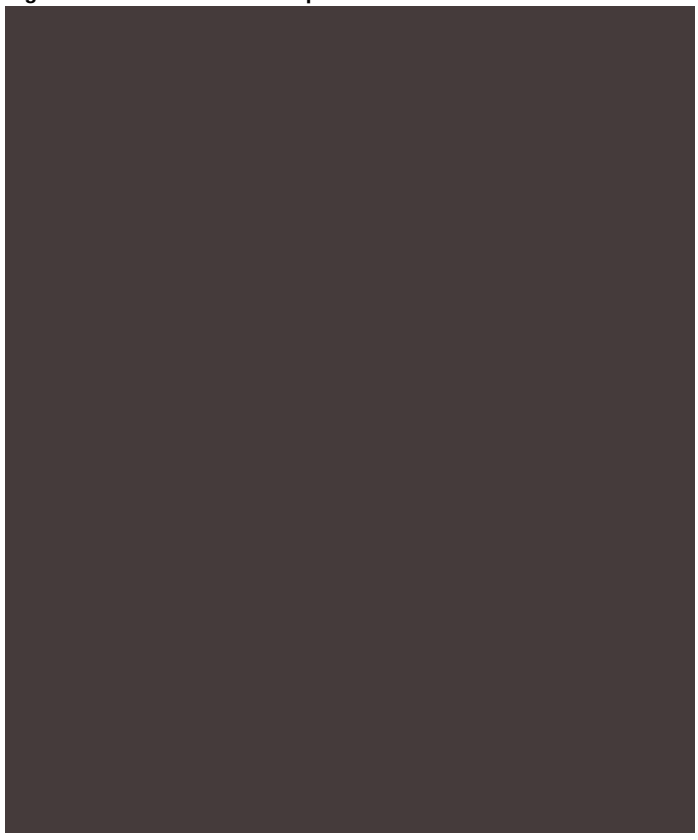
   On the Select components panel, select the `Typical installation`.
3. When the installation is complete, reboot your system and continue with the tutorial.

Step 2 - Create a CVS repository

Start the CVSNT control panel and create a repository for your OptimalJ projects.

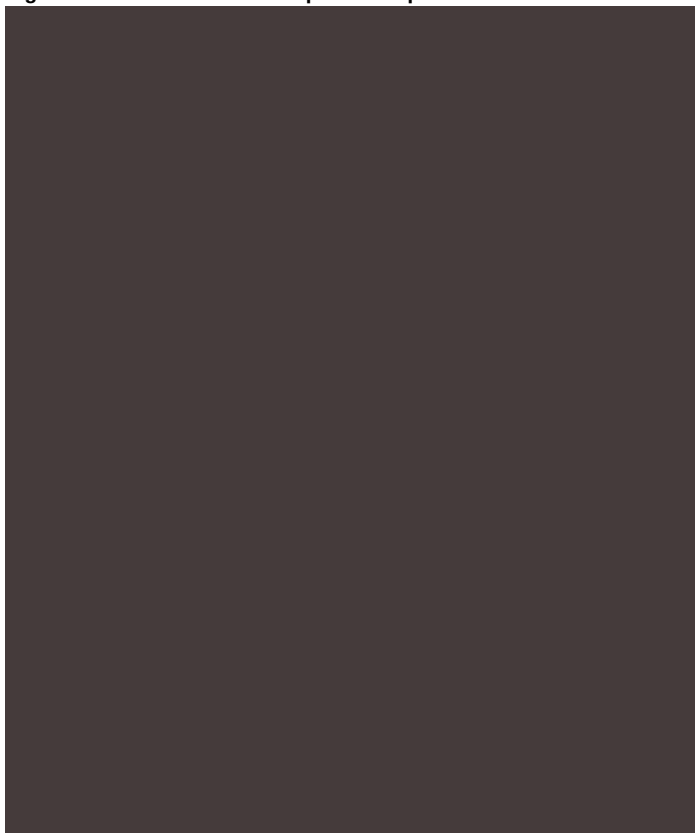
1. From the Windows task bar, choose **Start>Settings>Control Panel**. Open the CVSNT control panel.

Figure 1-129 CVSNT control panel - Service Status



2. Stop the CVS Service and the CVS Lock Service by clicking the Stop button for each service. Wait for the control panel to indicate that the services have stopped.
3. Click the Repositories tab.

Figure 1-130 CVSNT control panel - Repositories



4. Specify the root directory for all of your CVS repositories. Select the Repository Prefix checkbox and click the ellipsis button to select the root folder for your repository. Select the `C:\` drive and click New Folder to create the `C:\CVSRepository` folder.

Figure 1-131 Selecting a prefix folder

Select the new `C:\CVSRepository` folder and click OK.

5. Add a repository for OptimalJ projects. Click Add. Enter a path of `C:/CVSRepository/OJProjects` and click OK.

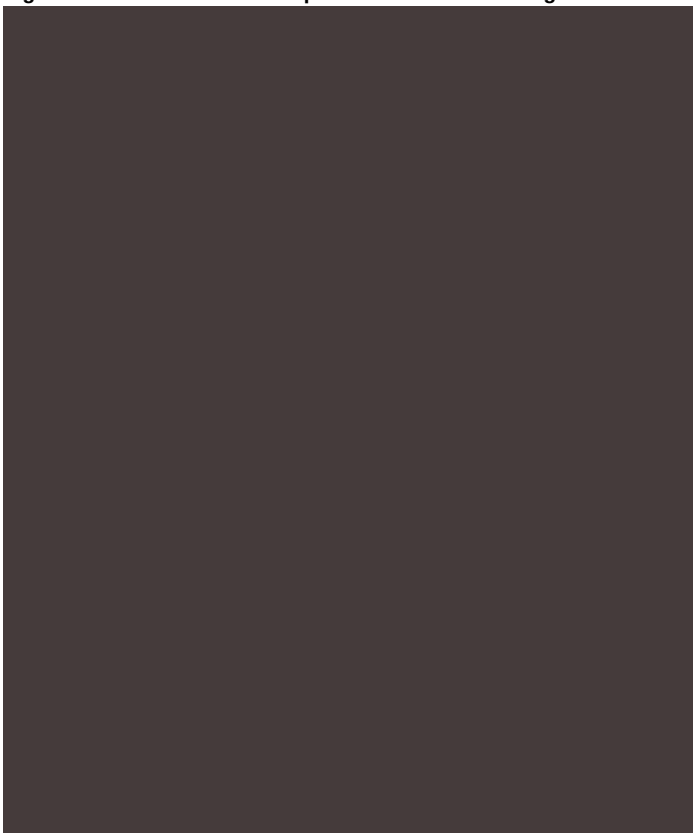
Note: CVSNT prefers forward slashes (/) for directory delimiters, even on Windows systems.

Figure 1-132 Creating an OJProjects root

CVSNT prompts for you to create the new directory. Click Yes. The new repository appears in the list of valid roots.

6. Set the configuration options for CVSNT. Click the Advanced tab.

Figure 1-133 CVSNT control panel - Advanced settings



Configure CVSNT as shown in the figure. The Temporary Directory must not be located in the Windows directory (for example, `C:\WINNT`) or anywhere in the `C:\Documents and Settings` directory. These directories have user permissions that can interfere with CVSNT.

7. Click Apply.
8. Click the Service Status tab. Start the CVS Service and the CVS Lock Service by clicking the Start button for each service. Wait for the control panel to indicate that the services are running.
9. Click OK to close the CVSNT control panel.

Step 3 - Add users for your repository

Your repository will require at least one user account so you can log in and interact with the repository. You will create two user accounts on this local computer so you can simulate OptimalJ development in a team environment.

1. On the Windows desktop, right-click the My Computer icon and choose **Manage**. Your system display the Computer Management window
2. In the Tree pane, browse to **Computer Management (Local)>System Tools>Local Users and Groups>Users**.

Figure 1-134 Computer Management - Users

3. Add an account for an OptimalJ architect. Right click in the users list and choose **New User**. Edit your New User settings as shown in the following figure. In the Password fields, enter `optimalj`.

Figure 1-135 Settings for OptimalJ architect account

4. Click Create to save you changes. The New User dialog box clears in preparation for adding another account.
5. Add an account for an OptimalJ architect. Edit your New User settings as shown in the following figure. In the Password fields, enter `optimalj`.

Figure 1-136 Settings for OptimalJ developer account

6. Click Create to save you changes. The New User dialog box clears in preparation for adding another account. Click Close to close the New User dialog box. The new accounts appear in the user list.

Note: The selected account names and settings used here are designed to accommodate the OptimalJ CVS tutorials and have no bearing on the roles of OptimalJ architects or developers. Your CVS administrator will set up the user accounts in your production environment.

Figure 1-137 Computer Management - Users

7. Close the Computer Management window.

Step 4 - Configure your repository for OptimalJ

Your CVS repository will need to be configured to handle the specific file types used in OptimalJ projects. Normally, the CVS administrator would check out and edit the repository configuration files because the configuration files in the repository are usually restricted to administrators only. Since we are configuring a test repository purely for demonstration purposes, we will perform the changes ourselves and manually edit the configuration files in the `C:\CVSRepository\OJProjects` directory.

In a production environment, your CVS administrator would handle this configuration for you, because the configuration files in the repository are usually restricted to administrators only. CVS administrators should see *Configuring a CVS repository for OptimalJ* for more information.

1. In Windows Explorer, open the `C:\CVSRepository\OJProjects\CVSROOT` directory.

2. Select the `cvswrappers` file. This file tells CVS which file extensions apply to binary files. Open the properties for the file, clear the Read Only checkbox, an click OK.
3. Use a text editor, such as WordPad, to add the following text to the end of the `cvswrappers` file.

```
1 | *.bmp -k 'b'
2 | *.gif -k 'b'
3 | *.ear -k 'b'
4 | *.idx -k 'b'
5 | *.jar -k 'b'
6 | *.jpeg -k 'b'
7 | *.jpg -k 'b'
8 | *.mcr -k 'b'
9 | *.png -k 'b'
10 | *.war -k 'b'
11 | *.wbmp -k 'b'
12 | *.xenon -k 'b'
13 | *.zip -k 'b'
```

Note: Your text editing program may try to append a `.txt` extension to the `cvswrappers` filename. If this happens, manually remove the extension in Windows Explorer.

4. Select the `cvswrappers` file. Open the properties for the file, select the Read Only checkbox, an click OK.
5. Use a text editor, such as WordPad, to create a new file named `cvsignore` in the `C:\CVSRepository\OJProjects\CVSROOT` directory. The `cvsignore` file should contain the following text.

```
1 | .nbattrs
2 | .nbattrs*
3 | *.*~
4 | *.class
5 | *.class*
6 | *.dar
7 | *.ear
8 | *.eardef
9 | *.jar
10 | *.mcr
```

```
11 | *.tpl
12 | *.war
13 | *.xenon
14 | *.zip
```

Note: Your text editing program may try to append a `.txt` *extension to the* `cvswrappers` *filename. If this happens, manually remove the extension in Windows Explorer.*

6. Select the `cvsignore` file. Open the properties for the file, select the Read Only checkbox, an click OK.

You successfully installed CVSNT, created a repository for OptimalJ projects, and configured that repository to properly handle OptimalJ projects. You added two user accounts for accessing the repository.

Your repository is now ready to import and manage OptimalJ projects. The *Working with OptimalJ projects in CVS* will show how you can manage OptimalJ projects with CVS.

Further reading

For more information, see the documentation under *Using version control in OptimalJ*.

The full documentation for CVSNT is located at *http://www.cvsnt.org/wiki/*.

1.30 Working with OptimalJ projects in CVS

In this tutorial, you import a project into a CVS repository. You will then simulate the activities of two OptimalJ users, each one checking out and editing the same project in the CVS repository. One user is the OptimalJ architect who will model the application and generate the code for the development team. The second user is a OptimalJ developer who is editing the generated code provided by the architect.

Prerequisites

- You should have already created a project in OptimalJ. The CVS tutorials are based on the sample CRM project that is included with OptimalJ. You can quickly generate this project by following the *Importing a domain class model* tutorial.

  Before you import, you should perform the following tasks on your application:

  - Update all models.
  - Generate all code.
  - Optionally, compile and test your project.

- You should have already perform the steps in the *Installing a local CVS server* tutorial.
- You should be comfortable with editing files in the Explorer [Code Model] and editing DomainAttributes.

Duration

This tutorial takes approximately two hours to complete.

Objectives

This tutorial will provide a view of typical editing workflow when using a CVS repository to store an OptimalJ project.

Step 1 - Prepare the file system

In your file system, create the following directories:

- `C:\CVSOJArch`?this directory will contain the local work area for the OptimalJ architect.
- `C:\CVSOJDev`?this directory will contain the local work area for the OptimalJ developer.

Step 2 - The architect imports the OptimalJ project into CVS

You are an OptimalJ architect and you have just finished the modeling and code generation work on your application. It is time to import the application into the CVS repository and make it available to the rest of the development team. Your application consists of multiple directories. Each directory will need to be imported into the CVS repository separately.

1. In OptimalJ, choose **Tools>Import module to CVS**. The Import Cvs Wizard appears.
2. Select the `C:\OptimalJ\umlImport\umlModel` directory and click Next.
3. Enter the connection method and settings for importing the selected directory.

Figure 1-138 Import Cvs Wizard?Connection settings

Enter the following values:

- **Server Name**?IP address or DNS name of the server. This value will be provided by your CVS administrator. For this tutorial, enter `localhost`.
- **Port**?Communications port for your CVS server. This value will be provided by your CVS administrator. For this tutorial, enter `2400`.
- **User Name**?Your user name on the CVS Server. Your CVS administrator creates this user name for you. We are currently playing the part of the OptimalJ architect, so enter `ojarchitect`.

- **Repository**?Path to your repository on the CVS server. This value will be provided by your CVS administrator and must start with a front slash (/). For this tutorial, enter `/OJProjects`.
- **CVSROOT=**?A read-only field that displays the equivalent CVS command that results from your current settings.

When you have entered your settings, click Next.

4. Provide the password for your user name.

Figure 1-139 Import Cvs Wizard?Client login

Enter the password for your account on the CVS server. For this tutorial, enter `optimalj` and click Login. The CVS server responds with "Login successful". Click Next.

5. Leave the import options set to their default values. The **Module Name** field contains the name of the module that will be created in the CVS repository. You will need the module name later when you checkout your project.

Figure 1-140 Import Cvs Wizard2 - Import options

Click Finish.

The output from the CVS server appears in the Output tab of the Output Window. When the import is complete, the status bar displays *Cvs Import finished successfully*.

6. Repeat the import process for the `C:\OptimalJ\umlImport\umlEjbCode` directory.

1. Choose **Tools>Import module to CVS**.
2. Select the `C:\OptimalJ\umlImport\umlEjbCode` directory and click Next.
3. Enter the same connection settings you used earlier in this step. Click Next.
4. Enter your `optimalj` password and click Login. Click Next.
5. Click Finish.

6. Repeat the import process for the `C:\OptimalJ\umlImport\umlWebCode` directory.

1. Choose **Tools>Import module to CVS**.
2. Select the `C:\OptimalJ\umlImport\umlWebCode` directory and click Next.
3. Enter the same connection settings you used earlier in this step. Click Next.
4. Enter your `optimalj` password and click Login. Click Next.
5. Click Finish.

All the models, EJB code and Web code for the `umlImport` application are now available in the CVS repository. To edit this project, you must check the application directories out and mount them in your working directory.

Step 3 - Developer checks out application from CVS

You are now playing the role of an OptimalJ developer. Your OptimalJ architect has made a new application available in the CVS repository and you want to edit the application. First you will need to check the application out from the CVS repository.

1. In OptimalJ, choose **Tools>Checkout module from CVS**. The Checkout Wizard appears.
2. Select the `C:\CVSOJDev` directory and click Next.
3. Enter the connection method and settings for performing the checkout.

Figure 1-141 Checkout Wizard?Connection settings

Enter the following values:

- **Server Name**? `localhost`.
- **Port**? `2400`.
- **User Name**? `ojdeveloper`.
- **Repository**? `/OJProjects`.

When you have entered your settings, click Next.

4. Provide the password for your user name.

Figure 1-142 Checkout Wizard2lient login

Enter the password for your account on the CVS server. For this tutorial, enter `optimalj` and click Login. The CVS server responds with "Login successful". Click Next.

5. Specify the modules that you would like to check out.

Figure 1-143 Checkout Wizard - Select modules

In the **Additional Modules for Checkout** field, enter `umlModel umlEjbCode umlWebCode`. This will checkout all three of the modules for the `umlImport` application. Click Next.

6. Click Finish.
7. The output from the CVS server appears in the Output tab of the Output Window. When the checkout is complete, the status bar displays *Cvs Checkout finished successfully*.

You have checked out a copy of the `umlImport` project into your working directory. You can now create a project and mount the checked out directories.

Step 4 - Architect checks out application from CVS

You are back to playing the role of the OptimalJ architect. You previously imported the application and now you need to check the application out from the CVS repository to make some changes.

This procedure is similar to the procedure in Step 3 - Developer checks out application from CVS.

1. In OptimalJ, choose **Tools>Checkout module from CVS**.
2. Select the `C:\CVSOJArch` directory and click Next.
3. Enter the following values:

- **Server Name**? `localhost`.
- **Port**? `2400`.
- **User Name**? `ojarchitect`.
- **Repository**? `/OJProjects`.

When you have entered your settings, click Next.

4. Enter a password of `optimalj` and click Login. The CVS server responds with "Login successful". Click Next.
5. In the **Additional Modules for Checkout** field, enter `umlModel umlEjbCode umlWebCode`. This will checkout all three of the modules for the `umlImport` application. Click Next.
6. Click Finish.
7. The output from the CVS server appears in the Output tab of the Output Window. When the checkout is complete, the status bar displays *Cvs Checkout finished successfully*.

Step 5 - Developer mounts project for editing

As the OptimalJ developer, you want to open the project you just checked out. You will need to create a project and mount the application directories.

1. Choose **Project>Project Manager**. Click New, enter a new project name of `CVSDev` and click OK. The new CVSDev project opens.
2. In the Explorer [Code Model], select the Code Model node. Right-click and choose **Mount>Version Control>CVS**. The New Wizard?CVS appears.
3. Click the Browse button and select the `C:\CVSOJDev\umlModel` directory and click Open. Click next.
4. OptimalJ knows that you checked out the selected directory, so it prompts you, specifically, for your CVS password.

Figure 1-144 New Wizard - CVS Client login

Enter `optimalj` and click Login. The CVS server responds with "Login successful".

Click Finish.

OptimalJ loads the model repository for your project.

5. Repeat the mount process for the `C:\CVSOJDev\umlEjbCode` directory.

1. Right-click the Code Model node and choose **Mount>Version Control>CVS**.
2. Select the `C:\CVSOJDev\umlEjbCode` directory. Click next.
3. Enter a password of `optimalj` and click Login. Click Finish.
4. Repeat the mount process for the `C:\CVSOJDev\umlWebCode` directory.

1. Right-click the Code Model node and choose **Mount>Version Control>CVS**.

2. Select the `C:\CVSOJDev\umlWebCode` directory. Click next.
3. Enter a password of `optimalj` and click Login. Click Finish.

Step 6 - Developer adds web module jar file

Since most project jar files (for example, `ejb.jar`) are derived from the source code, they are not included in the CVS repository. However, the `alturalibDeployWeb.jar` is required by your application at compile time and deploy time. Since your project was checked out from a CVS repository that, by rule, does not contain `.jar` files, you will need to manually restore this file to your application.

1. In the Explorer [Code Model], select the Code Model node. Right-click and choose **Mount>Archive Files**. The New Wizard? Archive Files appears.
2. Select the `OptimalJInstall\modules\ext\weblib.zip` file and click Finish. The contents of `weblib.zip` are mounted in the Explorer [Code Model].
3. In the Explorer [Code Model], right-click `weblib.zip\WEB-INF\lib\alturalibDeployWeb.jar` and choose Copy.
4. Right-click the `C:\CVSOJDev\umlWebCode\WEB-INF\lib` folder and choose **Paste>Copy**.

Figure 1-145 alturalibDeployWeb.jar added to your project

You can already see how the interface for a CVS file system different. Each filename in the Explorer [Code Model] is appended

with the file's CVS status and version number. Files that you've just checked out from the repository are up-to-date. The `alturalibDeployWeb.jar` exists in your working directory, but not on the CVS repository, so it has a status of Local.

5. Right-click the `weblib.zip` node and choose **Unmount Filesystem**.
6. Before you proceed to edit the project, it is wise to compile the project to make sure all the necessary components are checked out. Choose **Project>Compile Project**. OptimalJ displays the message of *Finished Project CVSDev* when compilation is complete.

Step 7 - Architect mounts project for editing

The OptimalJ also needs to mount their checked out project for editing. You will need to create a project and mount the application directories for the architect.

This procedure is similar to the one in Step 5 - Developer mounts project for editing.

1. Choose **Project>Project Manager**. Click New, enter a new project name of `CVSArch` and click OK.
2. In the Explorer [Code Model], select the Code Model node. Right-click and choose **Mount>Version Control>CVS**. The New Wizard?CVS appears.
3. Mount the `C:\CVSOJArch\umlModel` directory.

1. Right-click the Code Model node and choose **Mount>Version Control>CVS**.
2. Select the `C:\CVSOJArch\umlModel` directory. Click next.
3. Enter a password of `optimalj` and click Login. Click Finish.

4. Mount the `C:\CVSOJArch\umlEjbCode` directory.

1. Right-click the Code Model node and choose **Mount>Version Control>CVS**.
2. Select the `C:\CVSOJArch\umlEjbCode` directory. Click next.
3. Enter a password of `optimalj` and click Login. Click Finish.

4. Mount the `C:\CVSOJDev\umlWebCode` directory.

1. Right-click the Code Model node and choose **Mount>Version Control>CVS**.
2. Select the `C:\CVSOJDev\umlWebCode` directory. Click next.
3. Enter a password of `optimalj` and click Login. Click Finish.

Step 8 - Architect adds web module jar file

The architect will need to add the `alturalibDeployWeb.jar` just like the developer did in Step 6 - Developer adds web module jar file.

1. In the Explorer [Code Model], select the Code Model node. Right-click and choose **Mount>Archive Files**. The New Wizard? Archive Files appears.
2. Select the `OptimalJInstall\modules\ext\weblib.zip` file and click Finish. The contents of `weblib.zip` are mounted in the Explorer [Code Model].
3. In the Explorer [Code Model], right-click `weblib.zip\WEB-INF\lib\alturalibDeployWeb.jar` and choose Copy.
4. Right-click the `C:\CVSOJDev\umlWebCode\WEB-INF\lib` folder and choose **Paste>Copy**.
5. Right-click the `weblib.zip` node and choose **Unmount Filesystem**.
6. Choose **Project>Compile Project** to perform a test compile on the project. OptimalJ displays the message of *Finished Project CVSDev* when compilation is complete.

You can now simulate the actions of the architect and developer editing the same project. Open the *CVSArch* project to play the role of the architect. Open the *CVSDev* project to play the role of the developer.

Step 8 - Developer edits the CustomerMaintBrowse.jsp

As an OptimalJ developer, you want to add a *birthday* property to the project's `CustomerMaintBrowse.jsp` file.

1. Take the role of the developer. Choose **Project>Project Manager**. Click the `CVSDev` project and click Open. The CVSDev project opens.
2. In the Explorer [Code Model], open the `umlWebCode` folder.
3. Double-click the `CustomerMaintBrowse.jsp` file to edit it in the Source Editor.
4. Locate the free block that starts with the line:

   `<%-- add your own properties here --%>`

   Edit this free block so it looks like the following:

```
<%--  add your own properties here  --%>
property: birthday
```

5. Close this file and save your changes.

6. In the Explorer [Code Model], the status of\ `CustomerMaintBrowse.jsp` has changed from Up-to-date to LMod, indicating that the file has been modified locally.

Figure 1-146 Locally modified file

To make this change available to the rest of the development team, right-click the `CustomerMaintBrowse.jsp` and choose **CVS>Commit**.

7. OptimalJ displays a dialog box about advanced CVS options. Select the Do Not Show This Dialog Box Again checkbox and click Commit. The Arguments for Commit Command dialog box appears.

Figure 1-147

Enter a Log Message of `Added birthday property` and click OK.

8. The Output of CVS Commands [Update] window appears. The entry in this window indicates that the changes in this file have been incorporated into the CVS repository. The version of the `CustomerMaintBrowse.jsp` file has changed from 1.1.1.1 to 1.2.

Figure 1-148 Commit results for CustomerMaintBrowse.jsp

Click Close to dismiss the Output of CVS Commands [Update] window.

Step 9 - Architect adds class attribute

The architect has decided that the Customer domain class needs an additional attribute to store the middle initial of the customer. They will edit the domain class, update models, and generate code. This change will affect several files.

1. Take the role of the architect. Choose **Project>Project Manager**. Click the `CVSArch` project and click Open. The CVSArch project opens.

2. In the Explorer [Domain Model], browse to `mycrm.domain.class.customer`.
3. Right-click the Customer domain class and choose **New Child>DomainAttribute**. Add a `middleInit` attribute of type `String` and click Finish.
4. This change requires that you update all models. Choose **Model>Update All Models**. Select the `mycrm` node and click Finish. Wait for the model update to complete.
5. Generate all code. Choose **Model>Generate All Code**.

Note: The first time you generate code after you check out a project, you need to re-establish your project's EJB code and Web code folders.

6. When prompted define code directories, click Mount Each filesystem yourself and click OK.
7. When prompted for a directory for the EJB code, select the `umlEjbCode` file system and click OK.

   Wait for the generation process to complete before continuing.
8. When prompted for a directory for the web code, select the `umlWebCode` file system and click OK.

   Wait for the generation process to complete before continuing.
9. In the Explorer [Code Model] open the `umlWebCode` node and observe that all the `.jsp` files have been locally modified.
10. To incorporate all the code changes into the CVS repository, the architect must execute a Commit command. However, before you can perform a commit, you must check to see if any of the files in the CVS repository have been updated. To recursively refresh the CVS status for all files in the `umlWebCode` folder, right-click the `umlWebCode` folder and choose **CVS>Refresh Recursively**. The Explorer [Code Model] updates the CVS status for the files in the `umlWebCode` folder.

Figure 1-149

The Refresh command has revealed the following:

- Even though all the `.jsp` files had been marked as locally modified, the content of several files (for example, `MainMenu.jsp`) is no different than the content already in the CVS repository. This is why these files are now marked as Up-to-date.
- Several files have been modified and will need to be committed to the CVS repository.
- The `CustomerMaintBrowse.jsp` file, which had been modified by the developer, will need to be merged (via the Update command) before it can be committed to the CVS repository.

11. To recursively incorporate all file updates in the `umlWebCode` folder, right-click the `umlWebCode` node and choose **CVS>Update**.

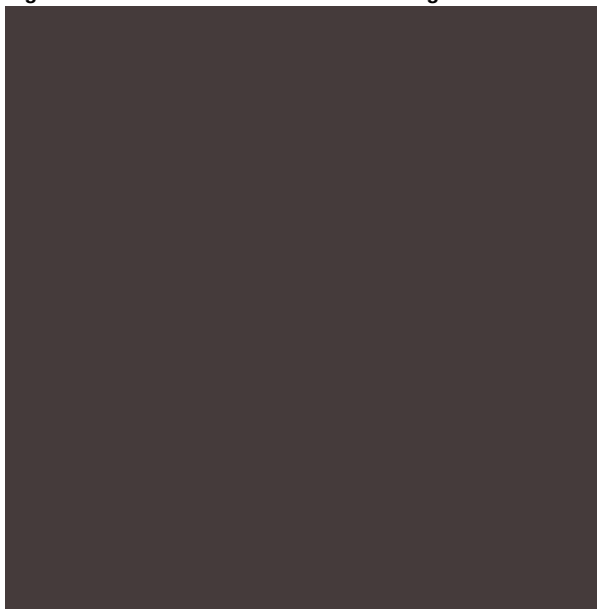
Figure 1-150 Output from Update command

The Output of VCS Commands [Update] window shows that several files are modified and need to be committed to the CVS repository. One file, `CustomerMaintBrowse.jsp` was merged. In the Explorer [Code Model] you can see that the version for `CustomerMaintBrowse.jsp` has increased to version 1.3 due to the merge operation.

Note: If the Type column for `CustomerMaintBrowse.jsp` showed a C, this would indicate that a merge conflict exists and need to be manually resolved.

12. Double-click `CustomerMaintBrowse.jsp` and verify that the developer's addition of the `birthday` property has been merged from the CVS repository. Close the Source Editor when you are done.
13. To recursively commit all changes in the `umlWebCode` folder, right-click the `umlWebCode` node and choose **CVS>Commit**.

Figure 1-151 Commit comment for adding middleInit



Enter a Log Message of `Added middleInit domain class` and click OK. The Output of VCS Commands [Commit] window logs the activity. Click Close to dismiss this window.

14. To incorporate the remaining changes into the CVS repository, commit the changes in the `umlEjbCode` and `umlModel` folders.

- Right-click the `umlEjbCode` node and choose **CVS>Commit**.
- Right-click the `umlModel` node and choose **CVS>Update**.
- Right-click the `umlModel` node and choose **CVS>Commit**.

Tip: When prompted for a Log Message, click Previous to recall the text from the last operation.

Step 10 - Developer updates project with middleInit change

Now the developer needs to incorporate the changes that the architect made.

1. Take the role of the developer. Choose **Project>Project Manager**. Click the `CVSDev` project and click Open. The CVSDev project opens.

2. In the Explorer [Code Model], open the `umlWebCode` folder and observe that `CustomerMaintBrowse.jsp` is still back at version 1.2.

Note: We already know that the architect has committed version 1.3 to the CVS repository.

3. To recursively incorporate updates from the CVS repository, right-click the `umlWebCode` node and choose **CVS>Update**. The Output of VCS Commands [Update] window logs the activity. Click Close to dismiss this window.
4. To incorporate the remaining changes from the CVS repository, update the `umlEjbCode` and `umlModel` folders.

- Right-click the `umlEjbCode` node and choose **CVS>Update**.
- Right-click the `umlModel` node and choose **CVS>Update**.

The developer's project is now synchronized with the architect's changes.

This tutorial has provided an overview of the typical setup and workflow associated with using a CVS repository with OptimalJ.

Further reading

General CVS documentation is available from the following locations:

- *http://www.gnu.org/manual/cvs-1.9/html_chapter/cvs_toc.html*.
- *http://www.cvshome.org/*

Refer to the *Using Version Control in the IDE* section in the *Core IDE Help* for more information on the CVS support in the NetBeans environment.