

Foundations of ATG Application Development Rel 10.1.2

Activity Guide

D81101GC11

Edition 1.1

October 2013

D84084

ORACLE®

Author

Karin Layher

Technical Contributors and Reviewers

Craig Canestrari

Kerwin Moy

Kevin Johnson

Rick Wilson

Editors

Raj Kumar

Smita Kommini

Publishers

Sumesh Koshy

Michael Sebastian Almeida

Syed Ali

Copyright © 2013 Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Table of Contents

Practices for Lesson 1: Essential Skills for Building on the ATG Platform	1-1
Practice 1-1: Creating a J2EE Web Application Module	1-2
Practices for Lesson 2: Using Components	2-1
Catching Up	2-2
Practice 2-1: Modifying Component Configurations	2-3
Practice 2-2: Creating a Component	2-4
Practices for Lesson 3: Dynamic Pages	3-1
Catching Up	3-2
Practice 3-1: Dynamusic Page Setup.....	3-3
Practice 3-2: Using Components in JSPs	3-4
Practice 3-3: Using Servlet Beans in Dynamic Pages.....	3-6
Practice 3-4: Using Page Fragments as Building Blocks	3-8
Practices for Lesson 4: Using Repositories	4-1
Catching Up	4-2
Practice 4-1: Working with Repositories.....	4-3
Practices for Lesson 5: Configuring SQL Repositories	5-1
Catching Up	5-2
Practice 5-1: Configuring and Using SQL Repositories.....	5-3
Practice 5-2: Creating Item Subtypes.....	5-11
Practices for Lesson 6: Using Repository Form Handlers	6-1
Catching Up	6-2
Practice 6-1: Using Repository Form Handlers	6-3
Practices for Lesson 7: Extending the Profile Repository	7-1
Catching Up	7-2
Practice 7-1: Extending the Profile and Using the Profile Form Handler	7-3
Practice 7-2: Using the Profile Form Handler	7-6

Ganesan Sree (ganesan186@gmail.com) has a non-transferable
license to use this Student Guide.

Practices for Lesson 1: Essential Skills for Building on the ATG Platform

Chapter 1

Practice 1-1: Creating a J2EE Web Application Module

Practice Objective

After completing this practice, you should be able to create a new ATG application and start it on WebLogic.

Problem Statement

Use Eclipse to create a new J2EE web application called `Dynamusic`. All lab work for this course will be contained in this application, a prototype of a music download site. To start, the application will consist of a set of static HTML “mockup” pages, which are provided for you.

Some of the practices in this document use and build upon baseline components that have been provided in a module called `Dynamusic-Base`. The module that you will create in this practice to contain your `Dynamusic` web application depends upon this base module. Do not worry too much about the contents of `Dynamusic-Base` at this time; this document describes these baseline components as needed to complete the individual practices.

The folders below will be referenced throughout these practices. The symbolic names listed will be used for quick and easy reference, and correspond to the designated path locations in the lab environment:

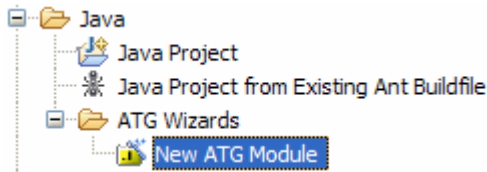
- **`atgroot`** : `/ae101`
The top-level directory where ATG is installed
- **`atgdir`** : `/ae101/ATG10.1.2`
Contains the installation of the ATG platform
- **`coursedir`** : `/ae101/Training/Foundations`
Contains the setup, source, and solution files for this course
- **`wldir`** : `/ae101/Middleware/wlserver`
Contains the WebLogic application server installation and the server configurations that will be used in class
- **`moduledir`** : `<atgdir>/Dynamusic`
The module containing the base code for the `Dynamusic` application that will be built in the labs for this course

Guided Instruction

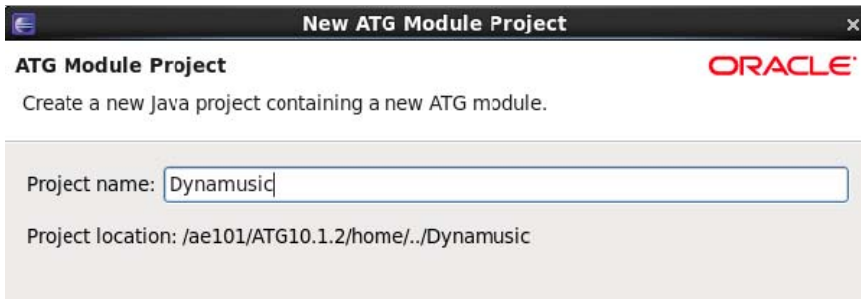
Create a New Application Module

1. Start Eclipse via the shortcut in the desktop folder called **FAD_Shortcuts**.
You can also start Eclipse by navigating to `atgroot/eclipse (/ae101/eclipse)` and invoking `eclipse.exe`.
2. Select File → New → Project.

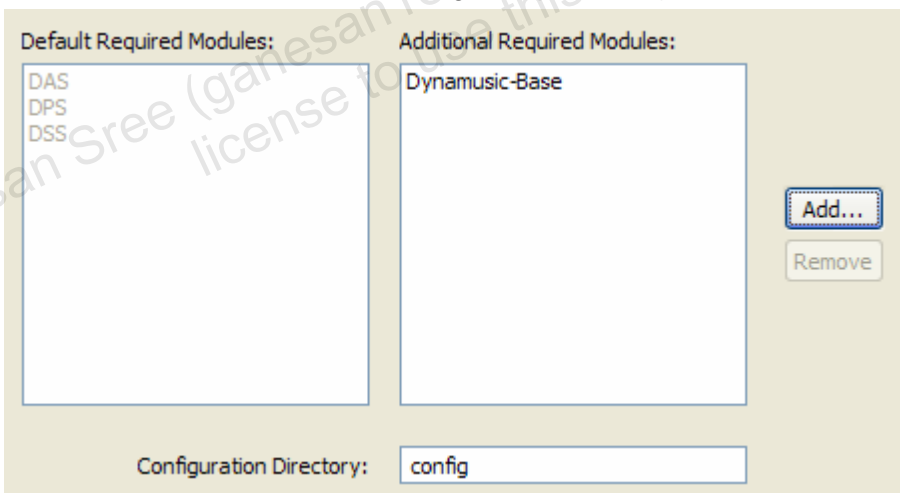
3. Select Java → ATG Wizards → New ATG Module and then click **Next**.



4. Enter project name `Dynamusic`, and make sure that the ATG root directory is correctly set for the location and version of the ATG installation (`/ae101/ATG10.1.2`). Then click **Next**.



5. When prompted for Application Stack, you do not need to add any modules for this course as the only module you will require (The Scenarios Module) is included by default. Click **Next**.
6. When prompted for additional required modules, click **Add** and then scroll down to choose `Dynamusic-Base` and click **OK**. The `Dynamusic-Base` module is provided as part of the Foundations lab setup to “jump start” some functionality for the module you are now creating. Leave the Configuration Directory as the default value `config`. (This will be discussed in the lesson titled “Using Components.”) Then click **Next**.



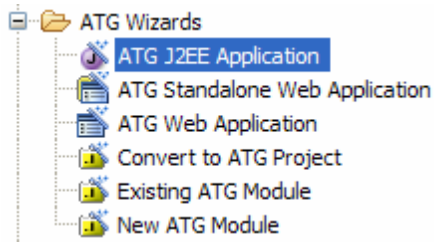
7. On the next screen (Java Settings), click **Finish**. You may see error messages at the bottom of the Eclipse window indicating that the build path is not set correctly. Don't worry about that for now, because you won't be building your own Java classes in this course.

If your new project does not appear automatically, click the Workbench icon to open the Eclipse Workbench.

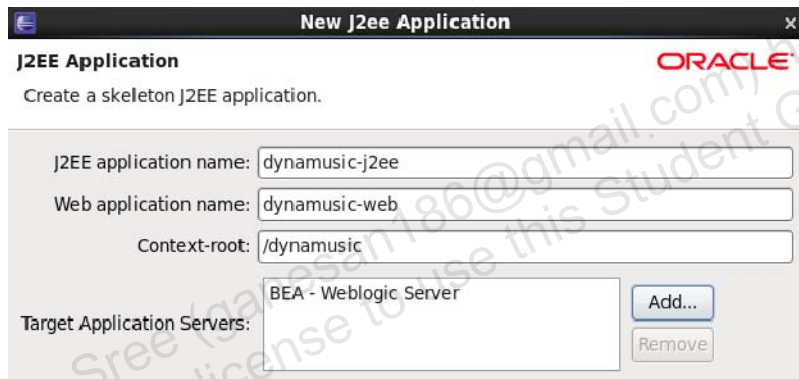


Create a New J2EE Application

1. Select **File** → **New** → **Other**.
2. Select **Java** → **ATG Wizards** → **ATG J2EE Application** and click **Next**.



3. Select the **Dynamusic** application module you just created and click **Next**.
4. Enter values as outlined below:
J2EE application name: `dynamusic-j2ee`
Web application name: `dynamusic-web`
Context-root: `/dynamusic`
For the target application server, click **Add** to choose BEA – Weblogic Server.



5. Click **Finish**.

Explore the Application Module

You will now familiarize yourself with the structure of your new module, both within Eclipse and by viewing the folders and files directly.

1. In Eclipse, view the Project Navigator (**Window** → **Show View** → **Navigator**). Open the Dynamusic project and explore the contents.
2. Double-click the `META-INF/MANIFEST.MF` file and edit the line for `ATG-Required` so that it looks like this:

`ATG-Required: Dynamusic-Base DafEar.Admin`

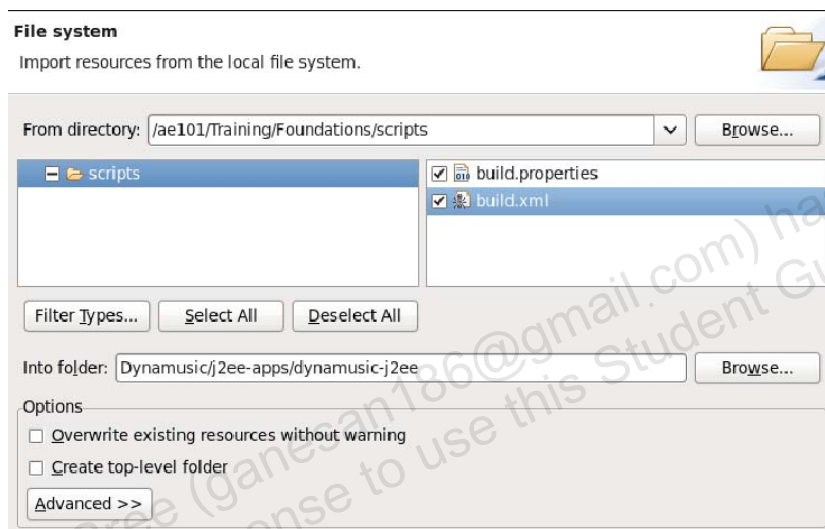
3. Save the file.
4. Using the Linux file browser, review the application structure created by the Eclipse plug-in (`atgdir/Dynamusic`).

Hint: Click on **Computer** → **Filesystem** from the desktop to access the file browser.

Import the Dynamusic Build Files

For this course, an Ant build script will be used to update files on a running instance of WebLogic Server. Ant is a tool for automating the software build process, implemented in Java and well-suited to building Java projects. The Ant script for this course has been specifically written to deploy, assemble and undeploy the Dynamusic application, and has been integrated into the course Eclipse development environment.

1. In Eclipse, in the Project Navigator, open the `Dynamusic` project and locate the `j2ee-apps/dynamusic-j2ee/` folder. Right-click the folder and select **Import**.
2. Under the **General** folder, select the import source to be **File System** and click **Next**.
3. For the source ("From") directory, browse to and select `atgroot/Training/Foundations/scripts/`. Click **OK**.
4. Select the two build files (`build.properties`, `build.xml`) and click **Finish**.

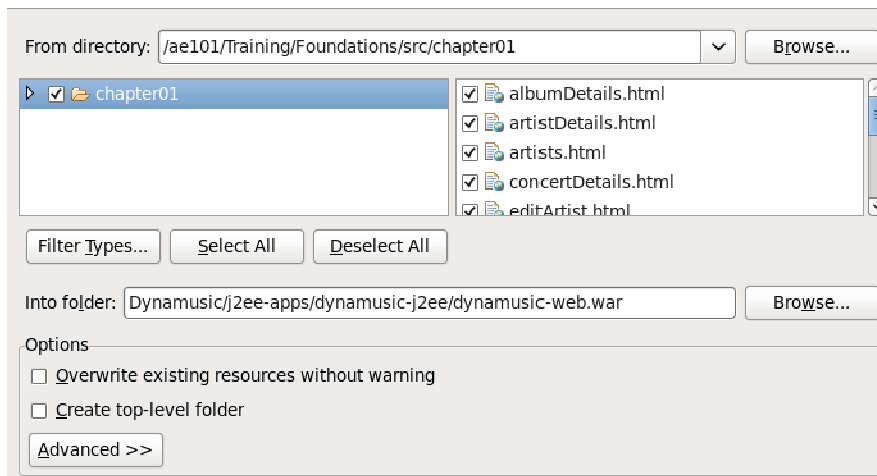


5. When prompted, click **Yes** to overwrite the existing file.

Import the Dynamusic Application Mockup Files into the J2EE Web Application

1. In Eclipse, in the Project Navigator, open the `Dynamusic` project and locate the `j2ee-apps/dynamusic-j2ee/dynamusic-web.war` folder. Right-click the folder and select **Import**.
2. Under the **General** folder, select the import source to be **File System** and click **Next**.
3. For the source ("From") directory, browse to and select `atgroot/Training/Foundations/src/chapter01/`. Click **OK**.

- Click the **Select All** button (or select the **chapter01** folder in the left pane) to copy all the contents (HTML files and folders).

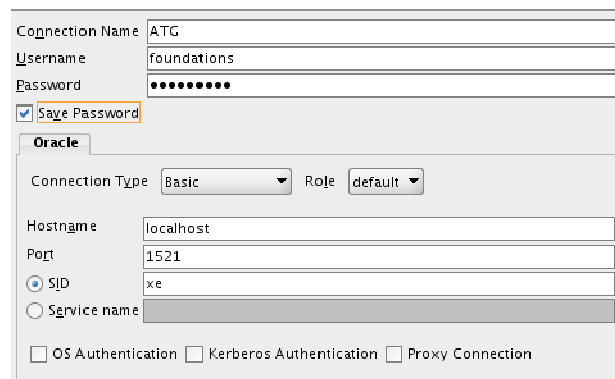


- Click **Finish**. Eclipse will copy the files and folders into the Dynamusic web application.

Examine the Data

- Oracle SQL Developer has been provided. To start it, open a terminal window and navigate to `/opt/sqldeveloper`. Run `./sqldeveloper.sh`. Leave this terminal window open.
- To create a new connection, dismiss the Tips pop-up if it appears, then:
 - Click the plus icon.
 - Enter the values provided below and click Connect.

Connection name: ATG
 Username: foundations
 Password: password1
 Hostname: localhost
 Port: 1521
 SID: xe



- On the Connections tab, select Connections, expand ATG, and expand the Tables section. The tables that have names beginning with `dynamusic_` are used for the custom repositories in the Dynamusic application. The other tables that you see (beginning with `das_`, `dps_`, `dss_`, and so on) are used by the ATG product modules that Dynamusic depends on (dependencies are specified in the `MANIFEST.MF` file of a given application module or product module).

ID	TITLE	COVER
1 300123	Purple Roses	images/PurpleRoses.jpg
2 300122	The Manhattan Album	images/TheManhattanAlbum.jpg
3 300121	Stardust	images/Stardust.jpg
4 300052	Runaway Train	images/RunawayTrain.jpg
5 300053	Welcome To The New World	images/WelcomeToTheNewWorld.jpg
6 300051	The Man From Texas	images/TheManFromTexas.jpg
7 300093	Bring On The Night	images/BringOnTheNight.jpg
8 300092	The Wolf	images/TheWolf.jpg

Start Your Servers

- From the `FAD_Shortcuts` folder on your desktop, double-click `startWLSAdmin`. When prompted, choose `Run in Terminal`. When the server is started, you should see something similar to:

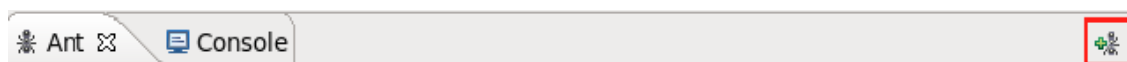
```
<Mar 26, 2013 3:09:03 PM EDT> <Notice> <WebLogicServer> <BEA-000331> <Started WebLogic Admin Server "AdminServer" for domain "atg_education" running in Development Mode>
<Mar 26, 2013 3:09:03 PM EDT> <Warning> <Server> <BEA-002611> <Hostname "localhost", maps to multiple IP addresses: 127.0.0.1, 0:0:0:0:0:0:0:1>
<Mar 26, 2013 3:09:03 PM EDT> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to RUNNING>
<Mar 26, 2013 3:09:03 PM EDT> <Notice> <WebLogicServer> <BEA-000360> <Server started in RUNNING mode>
```

- Now you need to start the ATG server. From the `FAD_Shortcuts` folder on your desktop, double-click `startATG`. When prompted, choose `Run in Terminal`. This server will take longer to start. You will know it's started when you see something similar to:

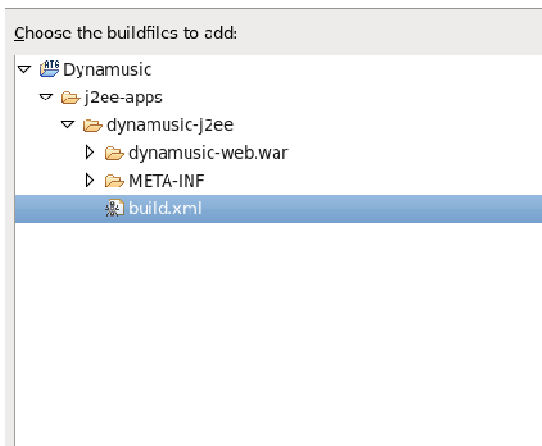
```
<Mar 26, 2013 3:32:01 PM EDT> <Notice> <Server> <BEA-002613> <Channel "Default[2]" is now listening on 0:0:0:0:0:0:0:1:7103 for protocols iiop, t3, ldap, snmp, http.>
<Mar 26, 2013 3:32:01 PM EDT> <Notice> <WebLogicServer> <BEA-000332> <Started WebLogic Managed Server "atg_production" for domain "atg_education" running in Development Mode>
<Mar 26, 2013 3:32:02 PM EDT> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to RUNNING>
<Mar 26, 2013 3:32:02 PM EDT> <Notice> <WebLogicServer> <BEA-000360> <Server started in RUNNING mode>
```

Deploy Your Application

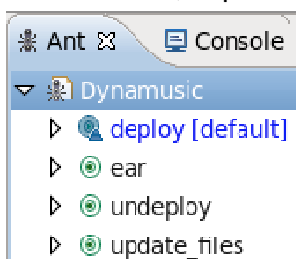
- In Eclipse, make sure that the Ant tab is viewable. (Window → Show View → Ant)
- Click the **Add Buildfiles** icon.



3. Navigate to the `build.xml` file under `dynamusic-j2ee` as seen below and click **OK**.



4. In the Ant tab, expand `Dynamusic` and you should see the following tasks:



5. Double-click the `deploy` task. This will actually run three of the tasks. It will first run the `undeploy` task that will remove the `Dynamusic` application from the ATG server (if there isn't one already deployed that is ok), then it will run the `ear` task which will assemble all the files and libraries needed to create the ear file that can be deployed to the server. Lastly, it will run the `deploy` task and deploy the ear file to the server. You can see its progress in the `Console` tab and you will know it has completed when you see something similar to:

```
[wldeploy] Task 1 initiated: [Deployer:149026]deploy application Dynamusic on atg_production.  
[wldeploy] Task 1 completed: [Deployer:149026]deploy application Dynamusic on atg_production.  
[wldeploy] Target state: deploy completed on Server atg_production  
[wldeploy] Target Assignments:  
[wldeploy] + Dynamusic atg_production  
[wldeploy] [ServerConnectionImpl.close():332] : Closing DM connection  
[wldeploy] [ServerConnectionImpl.close():352] : Unregistered all listeners  
[wldeploy] [ServerConnectionImpl.closeJMX():372] : Closed JMX connection  
[wldeploy] [ServerConnectionImpl.closeJMX():384] : Closed Runtime JMX connection  
[wldeploy] [ServerConnectionImpl.closeJMX():396] : Closed Edit JMX connection  
BUILD SUCCESSFUL  
Total time: 2 minutes 41 seconds
```

Test Your Application

1. In a browser, visit your web application's root page, which by default is a test page (`index.jsp`) auto-generated when you created your module in Eclipse:
`http://localhost:7103/dynamusic`. This page will confirm that the application is running correctly.
2. Go to `http://localhost:7103/dynamusic/home.html` and verify that the "mock-up" Dynamusic home page is loading correctly.
3. Explore the HTML version of Dynamusic. These hard-coded pages contain no dynamic content, but represent a "mock-up" of the final version of Dynamusic that you will be creating in class.
4. Navigate to the Admin UI in a browser: `http://localhost:7103/dyn/admin`. When prompted, log in as user `admin` (password `password1`). Bookmark this location for use throughout the rest of the course.

Ganesan Sree (ganesan186@gmail.com) has a non-transferable license to use this Student Guide.

Practices for Lesson 2: Using Components

Chapter 2

Catching Up

Successful completion of the practices for the lesson titled “Essential Skills for Building on the ATG Platform” is required to complete the practices in the rest of the course. It is essential that you have a working Dynamusic J2EE web application in place and that you are comfortable working in Eclipse in order to complete the remaining exercises. If you started but did not complete the exercises from the lesson titled “Essential Skills for Building on the ATG Platform,” continue from where you left off and complete the lab before proceeding. The instructor can assist you if you have run into any issues that are blocking successful completion of the practice.

If your web application is not working as intended and you would like to start over, go to the Eclipse Project View, right-click the Dynamusic project and select Delete. Select “yes” when asked if you want to delete the project files. This will reset you back to the default lab environment state and you can follow the guided instruction from the beginning to successfully build your Dynamusic web application.

Practice 2-1: Modifying Component Configurations

Practice Objective

After completing this practice, you will be able to modify the configuration of existing components.

Problem Statement

Set the default SMTP email “from” address to an application-specific value for Dynamusic.

Guided Instruction

Modify the `/atg/dynamo/service/SMTPEmail` component

1. View the ATG Component Browser in Eclipse (**Window → Show View → Other... → ATG → ATG Component Browser**).
2. In the component browser, navigate to and double-click the `/atg/dynamo/service/SMTPEmail` component. Change the configured value of the `defaultFrom` property to `customerservice@dynamusic.com` and click **Save**.
3. In the Eclipse project navigator, open `/Dynamusic/config/atg/dynamo/service/SMTPEmail.properties` and observe that your new property value has been recorded.
Note: You may need to refresh your project to see the change. To do so, right-click the Dynamusic project folder and select **Refresh**.
4. Run the “deploy” Ant task (remember to select **Window → Show View → Ant** if necessary to view the Ant tab), which will deploy your web application. Remember to wait for the “Build successful” log message in the Console window to verify the application was deployed successfully.
5. Go to the ATG Dynamo Admin UI by opening a browser to `http://localhost:7103/dyn/admin`.
6. Select the Component Browser link and browse to the `SMTPEmail` component (you will need the full Nucleus name, shown above).
7. Scroll down to the Properties table and verify that the live property value of `SMTPEmail.defaultFrom` is set to your modified value.

Practice 2-2: Creating a Component

Practice Objective

After completing this practice, you should be able to create components based on existing Java classes.

Problem Statement

Create a component to retrieve a list of featured songs from the Songs Repository (this repository is configured in the provided `Dynamusic-Base` module). A real-world application would probably use a targeter and a slot component, and business users would use the ATG Business Control Center (BCC) to set up the rules and triggering logic. Targeters, slots, and related functionality are covered in the *ATG Personalization and Scenario Development* training course.

This introductory practice uses a simplified approach. When instantiated, the `dynamusic.SongList` class will retrieve a list of featured song titles based on a hard-coded list (if no song IDs are configured, it will retrieve nine song titles by default). When `getSongs()` is called, it will return a subset of this list (based on settings in the `howMany` and `random` properties).

Guided Instruction

Create a New Component Based on a Custom Class

A custom class called `dynamusic.SongList`, which includes a `songs` property (a list of Strings), has been provided in the `Dynamusic-Base` module.

1. Open the File Browser and copy `coursedir/src/chapter02/FeaturedSongs.properties` to `atgdir/Dynamusic/config/dynamusic`. (You will need to create the `dynamusic` folder)
2. Double-click `FeaturedSongs.properties` to open it in an editor (choose Display when prompted).
3. Use the following table to set the values for your new component.

Property	Value
<code>\$class</code>	<code>dynamusic.SongList</code>
<code>\$scope</code>	<code>Global</code>
<code>howMany</code>	<code>4</code>
<code>itemDescriptor</code>	<code>Song</code>
<code>loggingDebug</code>	<code>True</code>
<code>random</code>	<code>False</code>
<code>repository</code>	<code>/dynamusic/SongsRepository</code>

4. **Save** your changes to the `/dynamusic/FeaturedSongs` component.

Start the Component and View Its Live Configuration Values

1. Navigate to the home page of the ATG Dynamo Admin UI (<http://localhost:7103/dyn/admin>). Click the Component Browser link and browse the `/dynamusic` configuration folder. The `FeaturedSongs` component is not displayed because it hasn't been started (you'll do that in the lesson titled "Dynamic Pages" by using it in a JSP). You can start it right now and view its properties by appending the component name to the URL for the `/dynamusic` folder in the Admin UI:

<http://localhost:7103/dyn/admin/nucleus/dynamusic/FeaturedSongs>

Click the `songs` property link to see the list of song titles.

2. Remember that the `FeaturedSongs` Nucleus component has global scope. This means that after the component is started, it stays live as long as the server keeps running (there is a way to shut down a live instance in the ACC, but that is only intended for developer testing). Live property values on global components can be changed by your application or by developer tools.
Click the link for the component name (`FeaturedSongs`) at the top of the page. Click the `howMany` property. Enter the number 3 into the New Value box and click the **Change Value** button. Now click `FeaturedSongs` again (at the top of the page), and then click the `songs` property to once again see the list of song titles. Scroll down and you should now see only three song titles listed. By default, the component loads nine songs as featured songs (stored internally), but the number returned in the `songs` property is controlled by the `howMany` property.
3. In the next practice, you will be writing logic into the home page to make a call to display the `FeaturedSongs.songs` component property. After that page is working successfully, you will reconfigure the `howMany` and `random` properties and test the results.

Ganesan Sree (ganesan186@gmail.com) has a non-transferable license to use this Student Guide.

Practices for Lesson 3: Dynamic Pages

Chapter 3

Catching Up

If you were unable to complete the practices from the lesson titled “Using Components,” perform the following steps:

1. Copy the `FeaturedSongs.properties` file from the `atgroot/Training/Foundations/solutions/chapter02` directory to `atgdir/Dynamusic/config/dynamusic/`. You will need to create the `dynamusic` folder.
2. In the Eclipse Navigator, refresh your Dynamusic project by right-clicking the Dynamusic folder and click **Refresh**.

Practice 3-1: Dynamusic Page Setup

Practice Objective

After completing this practice, you should be able to set the default (welcome) page of an ATG web application.

Problem Statement

Copy in the mockup JSPs from the course source folder, and set the Dynamusic web application's default page to `home.jsp`.

Guided Instruction

1. Import the contents of the folder `atgroot/Training/Foundations/src/chapter03` into the `dynamusic-web` web application. For a reminder on how to do this, refer back to the instructions for importing the Dynamusic application mockup files in the practice for lesson 1.
For the remainder of the course, you will be modifying the JSP mockup pages; to avoid confusion you may want to delete the HTML pages in the web application.
2. In Eclipse, double-click `dynamusic-web.war/WEB-INF/web.xml` to edit it. Open the `web-app` folder and scroll down to find the `welcome-file-list`. Change the value of the `welcome-file` setting, replacing `index.jsp` with `home.jsp`. **Note:** You can do this either in the Design or the Source views of the file.
3. **Optional:** The `web.xml` file you just edited uses the Servlet 2.3 spec, in which Expression Language (EL) is disabled by default. If you would like to use the Servlet 2.4 spec—in which EL is enabled by default—copy `atgroot/Training/Foundations/solutions/chapter03/web.xml` to `dynamusic-web.war/WEB-INF/web.xml`. This solution file already has the welcome file set to `home.jsp`, so the editing you did in the last step does not need to be done again.
4. Close the `web.xml` window and save your changes.
5. Run the Ant task `deploy` to redeploy your web application.

Practice 3-2: Using Components in JSPs

Practice Objective

After completing this practice, you will be able to display the properties of a Nucleus component in a JavaServer page (JSP).

Problem Statement

Create a personalized Dynamusic home page that displays a personalized welcome message using the `Profile` component.

You may use Eclipse, the ACC, or a text editor to view and modify JSPs. These practices assume you are using Eclipse.

Guided Instruction

Modify `home.jsp` to Welcome the User by Name

1. In the Eclipse project navigator panel, browse to `/Dynamusic/j2ee-apps/dynamusic-j2ee/dynamusic-web.war`, and double-click `home.jsp` to edit it.
2. Declare the DSP tag library at the top of the page, using the `taglib` directive. Note that this line must be the *first* non-comment line in the JSP.

```
<%@ taglib uri="/dspTaglib" prefix="dsp" %>
```
3. Enclose the entire page under the `taglib` declaration within `<dsp:page> ...`
`</dsp:page>` tags.
4. Locate the welcome message ("Welcome Arthur") and replace the static name with the user's name using a `<dsp:valueof>` tag. Get the value from the bean `/atg/userprofiling/Profile` (use the `firstName` property). Save your page.
5. Run the `update_files` Ant task to update your work on WebLogic server.
Note that this Ant task is different from the one you ran in previous practices. This script copies only the modified JSPs into the running Dynamusic rather than a full redeployment of the application. This task does not include a server restart, and completes much quicker than a full deployment. In many of the upcoming practices, you will only be required to deploy updated JSPs and will use this task to efficiently make the pages active.
6. To test your work, you must first log in to the application. Some existing test users are available for your use. Go to `http://localhost:7103/dynamusic/login.jsp` and log in as the user `ivan` (password `password1`). Verify that your customized welcome message is displayed.

Welcome, Ivan

This week's featured songs are:

7. You can look up additional test users in the ACC. If you have not already logged into the ACC, do so now. You can access the ACC via the Dynamo Administration UI (<http://localhost:7103/dyn/admin>) in your browser. Click the **ATG Control Center Administration** link, and then click the **Start ACC in Server VM** button. Log in to the ACC with user name / password of *admin / password1* and click the **Connect** button. To view existing test users, select **People and Organizations → Users** from the ACC's menu. Click the **List** button to see the currently defined users. (All the default users have the password *password1*.)

In order to test using a different user, enter the URL for `logout.jsp`. Confirm the logout, and then manually navigate again to `login.jsp` to log in as the next user. Note that the home page Logout link will not work as it is linked to `logout.html` rather than `logout.jsp`. You will fix this in a later exercise in this lesson.

Ganesan Sree (ganesan186@gmail.com) has a non-transferable license to use this Student Guide.

Practice 3-3: Using Servlet Beans in Dynamic Pages

Practice Objective

After completing this practice, you should be able use servlet beans (droplets) in JavaServer pages (JSPs).

Problem Statement

Modify the Dynamusic home page to display the custom welcome message only if the user is logged in. In addition, display a dynamic list of featured songs.

Guided Instruction

When displaying the welcome message, it's a good idea to add an additional check to first make sure that the user is logged in. One way to accomplish this is to use a `Switch` servlet bean to check the `Profile` object's `transient` property. The `transient` property will be `true` if the user is NOT logged in.

Note: You have several options for coding in JavaServer Pages. You can use the ACC and its wizards, manually write the code, or use the JSP editor in Eclipse along with the provided ATG snippets. The practices assume the use of the ATG snippets in Eclipse.

1. Edit `home.jsp` in Eclipse. Add a line to import the `/atg/userprofiling/Profile` bean (component) below the opening `<dsp:page>` tag.
2. In Eclipse, you will use an ATG snippet to insert the `Switch` droplet. To access the snippets in Eclipse, navigate to **Window → Show View → Other**. In the Show View window, select **Snippets** from the **General** menu and click **OK**. In the Snippets window, open the ATG Servlet Beans menu.
3. Edit `home.jsp`. Place your cursor on a new line in the area where the welcome greeting is displayed. Click the ATG Servlet Beans snippet menu once, scroll down, and double-click `Switch`. An input screen for the droplet will appear.
4. For the value input param, enter `Profile.transient`. Press Enter to accept the value. It will appear in the code window below. Click the **Insert Snippet** button to insert the code into the page. Editing in the page itself, change the `"value="` to `"bean="`, because you are accessing a `JavaBean` instead of a static value. Add `oparams` to handle the true and false cases. If `true`, indicating that the user has not logged in yet, display a generic greeting. If `false`, display the user's name using a `<dsp:valueof>` tag. You may need to move your welcome statement around to accommodate the droplet. Save your changes.
5. Run the `update_files` Ant task to update your work on WebLogic server.
6. Test your page as you did above, by navigating to the provided login page (`http://localhost:7103/dynamusic/login.jsp`) and logging in by using a test user profile (such as `user donna/password password1`). Also test if the page works if the user is not logged in, by navigating explicitly to the provided logout page (`http://localhost:7103/dynamusic/logout.jsp`). You will need to click the **Logout** button. Note that both of these pages redirect to the Dynamusic home page. Another alternative for the case of no one logged in is to bring up a new browser instance and view the home page directly (`http://localhost:7103/dynamusic/home.jsp`).

Dynamically Display Multivalued Component Properties

You are now going to replace the static “Featured Song” values on your home page to use the dynamic values you configured in a previous practice. Because the `songs` property of the `FeaturedSongs` component is multivalued, you will need to use a `ForEach` servlet bean to loop through the list and display each song.

1. Edit `home.jsp` in Eclipse and import the `/dynamusic/FeaturedSongs` bean (component) below the opening `<dsp:page>` tag.
2. Move your cursor to the area below your `Switch` droplet. Delete the static list of featured songs. Now insert a `ForEach` Servlet bean from the ATG Servlet Beans snippet menu. Loop over `FeaturedSongs.songs` (a list of `Strings`), displaying each song title in a formatted list. Here again, you will need to change the input parameter’s “value=” to “bean=”. Save your changes.
3. Run the `update_files` Ant task to update your work on WebLogic server.
4. Test your work by loading the Dynamusic home page from your browser. Note that if you modify, save and deploy a JSP, you can simply reload it from your browser to test your changes. When your page is working properly, reload the page several times to see the effect of `FeaturedSongs.random` being set to `false` (the default setting).
5. Navigate to the ATG Dynamo Admin UI (<http://localhost:7103/dyn/admin>) and click the Component Browser link to navigate to the `/dynamusic/FeaturedSongs` component. Under the Properties section, click and change the `howMany` property to a different number (from 1 to 9), and also set the `random` property to `true`. Reload the page several times to see the effect of these changes. Note these changes are only on the live component instance, not the `.properties` file.

Practice 3-4: Using Page Fragments as Building Blocks

Practice Objective

After completing this practice, you should be able to include a page fragment within a JSP, pass a parameter value to the fragment, and use that parameter within the fragment.

Problem Statement

Change the Dynamusic home page to use a common sidebar menu and page header. Pass the page header fragment the name of the page.

Guided Instruction

Use Page Fragments to Display a Common Sidebar Menu and Page Header

You will now add a standard sidebar to your site which will provide quick navigation links to the major Dynamusic functional pages. You will also add a standard header to the top of your home page. Because this header will be used by multiple pages in your site, you will want to dynamically display the name of the current page (Home Page, Profile Page, and so on) by passing in a parameter from the parent page.

1. Edit `common/sidebar.jsp` and change all the page links to point to the JSP (vs. HTML) version of each page. Declare the DSP tag library and enclose the page in a `dsp:page` tag.
2. Modify `home.jsp` to include `sidebar.jsp` in place of the individual html page links. (Comments will direct you to which portions to replace.)
3. Run the `update_files` Ant task and then browse to `http://localhost:7103/dynamusic` to test the page.
4. Modify `common/header.jsp`, replacing the static "Home" string with a `dsp:valueof` tag displaying the value of a parameter called `pagename`. Make sure you declare the DSP tag library and enclose the page in a `dsp:page` tag.
5. In `home.jsp`, use a `<dsp:include>` tag to include the `common/header.jsp` page. (Follow the comments for which portions to replace.)
6. Add a `<dsp:param>` tag to the `<dsp:include>` tag inserted in the previous step. The `param` tag should pass a parameter called `pagename` to the header page with a value set to the name of the page currently being displayed, such as "Home Page".
7. Run the `update_files` Ant task to update your work on WebLogic server and verify your changes.

Practices for Lesson 4: Using Repositories

Chapter 4

Catching Up

Completion of previous lessons is necessary in order to continue these practices. If you were not able to complete the practices from the lesson titled “Dynamic Pages,” perform the following:

1. Complete the “Catching Up” instructions from Practice 3, if you did not complete those practices.
2. Copy all files in `atgroot/Training/Foundations/src/chapter03` to `atgdir/Dynamusic/j2ee-apps/dynamusic-j2ee/dynamusic-web.war`.
3. Copy the `home.jsp` file from `coursedir/solutions/chapter03` to `atgdir/Dynamusic/j2ee-apps/dynamusic-j2ee/dynamusic-web.war`.
4. Copy the `header.jsp` and `sidebar.jsp` files from `coursedir/solutions/chapter03` to `atgdir/Dynamusic/j2ee-apps/dynamusic-j2ee/dynamusic-web.war/common`.
5. Copy the `web.xml` file from `coursedir/solutions/chapter03` to `atgdir/Dynamusic/j2ee-apps/dynamusic-j2ee/dynamusic-web.war/WEB-INF`.
6. Run the `update_files` Ant task.
7. In the Eclipse Navigator, refresh your Dynamusic project by right-clicking the Dynamusic folder and selecting **Refresh**.

Practice 4-1: Working with Repositories

Practice Objective

After completing this practice, you should be able to display repository data on Java Server Pages using standard servlet beans.

Problem Statement

Create pages that display a list of artists from the Songs Repository, the artist details for a particular artist (including a list of the artist's albums), and the album details for a particular album (including a list of the album's songs). A pre-populated Songs Repository has been provided for your use. The Songs Repository contains three item types: `song`, `album`, and `artist`.

There are four pages involved in this “drill-down” section for the Songs Repository. You will create the first three shown below; the fourth is provided for you:

artists.jsp: Lists the name of all the artists in the repository; each artist links to:

artistDetails.jsp: Shows details about the artist selected, and a list of albums by that artist; each album links to:

albumDetails.jsp: Shows details about the album select, and a list of songs on that album; each song links to:

song.jsp: Shows details about the song, including a link to download the song

Each page in this sequence links to the next by passing an `itemId` parameter, containing the ID of the selected item. For example, `artists.jsp` passes the ID of the selected artist to the `artistDetails.jsp` page. Refer to the lecture notes for a reminder of the details on how to implement this “drill-down” technique.

For these practices, and for most of the remaining practices in this course, JSP starter pages have been provided containing static elements that you will replace with dynamic ATG elements. These pages are provided to save you some development time. If you prefer, you can also use an empty “template” page that has been provided for you, called `dynamusic-template.jsp` (find it under `src/chapter03`). You can open this page, and then select **File** → **Save As** to save it as a new page you create. The area in which to insert the practice content is commented. The instructions below assume that you will be modifying the starter pages, but choose whichever method you find the most straightforward.


Guided Instruction

Explore the Songs Repository

1. Start the ACC if it is not currently running. Navigate to the Dynamo Admin UI (<http://localhost:7103/dyn/admin>), log in as user `admin` (password `password1`), click **ATG Control Center Administration**, and then click **Start ACC in Server VM**. Log in to the ACC as user `admin` (password `password1`).


Note that at times the ACC may time out, disconnecting from the ATG server, but still running in the server's VM. If that happens, choosing **Tools** → **Connect to server** may allow you to successfully log in again.

2. In the ACC, select **Content → SongsRepository**. Spend a few minutes becoming familiar with the `song`, `album`, and `artist` item types by performing the following tasks:

- List items of type `artist`.
- Select an `artist` and examine the properties.
- List items of type `album`.
- Select an `album` and examine its properties.
- List items of type `song`.
- Select a `song` and examine its properties.
- Use the query icon  to perform a complex query on the repository. Example:
items of type `song` whose `artist : name` starts with `T`
- Clear the query by clicking `whose` and selecting any `item`.

Additionally, you can open the `atgdir/Dynamusic-Base/config/dynamusic/songs.xml` file to review the property names for the `song`, `album`, and `artist` item types.

Create a Dynamic List of Artists

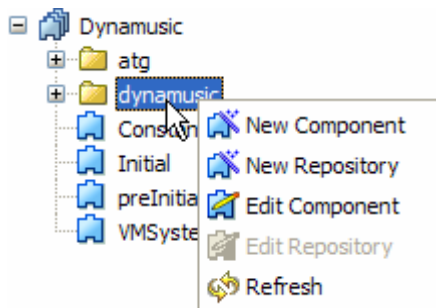
1. Edit `artists.jsp` (or create a new one using the `dynamusic-template.jsp` page, in your `dynamusic-web.war` directory). We suggest using Eclipse.
2. Locate the static list of artists. Delete the static list and replace it with a call to the `/atg/dynamo/droplet/RQLQueryForEach` servlet bean that will query for and display all the artist items in the repository. (If you are using Eclipse, you can use the appropriate snippet from the ATG Servlet Bean snippet list; if you are using the ACC to edit pages, you can use the Insert Servlet Bean  Wizard.) Set the input properties of the droplet as follows:

```
repository: /dynamusic/SongsRepository
itemDescriptor: artist
queryRQL: ALL ORDER BY name
```

3. In the `output` oparam, display the name of each artist using the `element` output parameter. In the `empty` oparam, which is rendered if there are no items, display a message to that effect.
4. Run the `update_files` Ant task.
5. Load the Dynamusic home page, select **Artists** and verify that all the artists from the Songs Repository are displayed.
6. Modify your artist list to display each artist name as a hyperlink to `artistDetails.jsp`, passing it a parameter called `itemId`, set to the `id` of the artist. You will work on `artistDetails.jsp` in a moment.
7. Load the Dynamusic home page, select **Artists** and verify that the names of the artists are now displayed as links to `artistDetails.jsp`.

Create a Repository-Based Artist Details Page

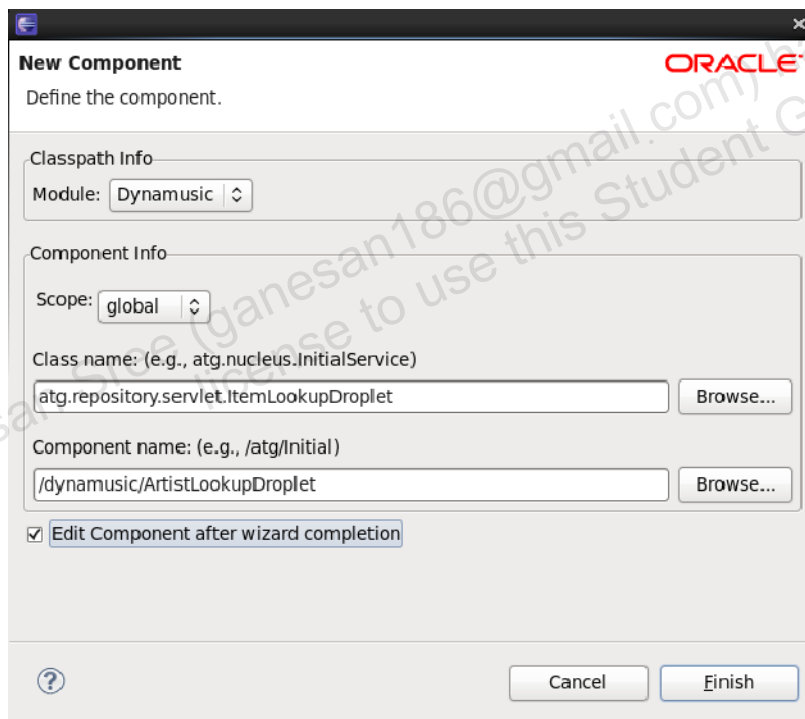
1. In the ATG Component Browser in Eclipse, right-click the **dynamusic** folder (not the Dynamusic project) from the configuration hierarchy and click **New Component**. (Make sure you are on the ATG Component Browser tab.)



2. Specify the appropriate values for the component (as shown below), select **Edit Component after wizard completion** and click **Finish**.

Class name: `atg.repository.servlet.ItemLookupDroplet`

Component name: `/dynamusic/ArtistLookupDroplet`



3. Set the `itemDescriptor` property to `artist` and link the `repository` property to `/dynamusic/SongsRepository`. Select **Save**.
4. Edit `artistDetails.jsp`. Use your `ArtistLookupDroplet` to replace the static artist information with artist data from the Songs Repository. Because you have configured `ArtistLookupDroplet` to query the `SongsRepository` and return artist repository items, the only thing missing is the repository ID of the artist. Remember the `itemId` parameter passed in from `artists.jsp`? It contains the `repositoryId` your servlet bean will use to fetch that artist. In the call to `ArtistLookupDroplet`, configure the `id` input parameter to use the `itemId` parameter passed in from `artists.jsp`. In the output `oparam`, you can display the name, `photoURL` (using `<img`

`src="<dsp:valueof param='...' />"`; note alternating nested quotation marks), and `description` properties.

You'll also need to pass the artist name to `header.jsp` so that it is displayed on the top of the page. So make sure that your `ArtistLookupDroplet` starts at the top of the JSP, before the inclusion of the header JSP fragment. Essentially, the entire body of the page will fall within the `output` oparam of the servlet bean. Unlike on other pages, the value you are passing to `header.jsp` is a dynamic value rather than a literal value.

5. Run the `update_files` Ant task. Load the **Artists** page. Select an artist and verify that the artist details (name, image, and description) are displayed correctly. (The albums displayed will still be static content; you will change this in the next step.)
6. Within your `ArtistLookupDroplet`, display the albums for the given artist using a nested `RQLQueryForEach` servlet bean. Remember that album item types have an `artist` property, so your query will need to look for all albums given an `artist.id`. That is, your `queryRQL` should indicate `artist.id = id of current artist passed in`. Review the lecture notes for a reminder of how to implement a parameterized query. For each album returned by the query, display the `title` property of the album.
Note: This is your first use of a nested droplet in the practices. You will notice that both the outer droplet (`ArtistLookupDroplet`) and the inner droplet (`RQLQueryForEach`) have an `element` output parameter. This overlap does not present a conflict. Within the inner droplet's `output` oparam, `element` will refer to an item returned by the inner droplet. If for some reason, you needed access to the outer droplet's `element` parameter, you should set a new parameter to its value before calling the inner droplet.
7. Run the `update_files` Ant task.
8. Reload the page (refresh in the browser). Select an artist and verify that the list of albums is displayed correctly.
9. Edit your album list and, for each album, display its `title` property as a hyperlink. The target page of the link will be `albumDetails.jsp`, and you will use the same technique as you did in `artistDetails.jsp`, passing a parameter called `itemId` representing the ID of the album.
10. Run the `update_files` Ant task and then reload the page and make sure each album name is a link passing the correct ID.

Create a Repository-Based Album Details Page

1. Create a new component in the `dynamusic` directory called `AlbumLookupDroplet` based on the `atg.repository.servlet.ItemLookupDroplet` class. This time, set the `itemDescriptor` property to `album`, and link the `repository` property to `/dynamusic/SongsRepository`.
2. Edit `albumDetails.jsp`. Use your `AlbumLookupDroplet` to replace the static album information with album data from the Songs Repository. Pass the album `title` to `header.jsp`. Display the `coverURL`, `description`, and `date` on the page.
3. Run the `update_files` Ant task. Load an artist detail page, select an album and verify your changes.

4. Display a list of songs from the album. Note that the `songList` property is multivalued (containing a set of `song` item types) and will need a nested `ForEach` servlet bean for display. Unlike with displaying an artist's albums, you will *not* need an RQL query droplet to do this; the query is implicit in the property definition for `songList`. Display the title of each song as a hyperlink, with the target page set to `song.jsp` (provided for you). You will need to pass a parameter called `itemId` containing the `id` of the `song` item to `song.jsp`.
5. Run the `update_files` Ant task. Reload the page and verify your changes.

Ganesan Sree (ganesan186@gmail.com) has a non-transferable license to use this Student Guide.

Ganesan Sree (ganesan186@gmail.com) has a non-transferable license to use this Student Guide.

Practices for Lesson 5: Configuring SQL Repositories

Chapter 5

Catching Up

If you were unable to complete the practices from the lesson titled “Using ATG Repositories,” perform the following steps:

1. Complete the “Catching Up” instructions from Practice 4, if you did not complete those practices.
2. Copy the properties files in *coursedir/solutions/chapter04* to *atgdir/Dynamusic/config/dynamusic*.
3. Copy the JSPs in *coursedir/solutions/chapter04* to *atgdir/Dynamusic/j2ee-apps/dynamusic-j2ee/dynamusic-web.war*.
4. In the Eclipse Navigator, refresh your Dynamusic project by right-clicking the Dynamusic folder and selecting **Refresh**.
5. Run the `update_files` Ant task to upload the latest files to the server.

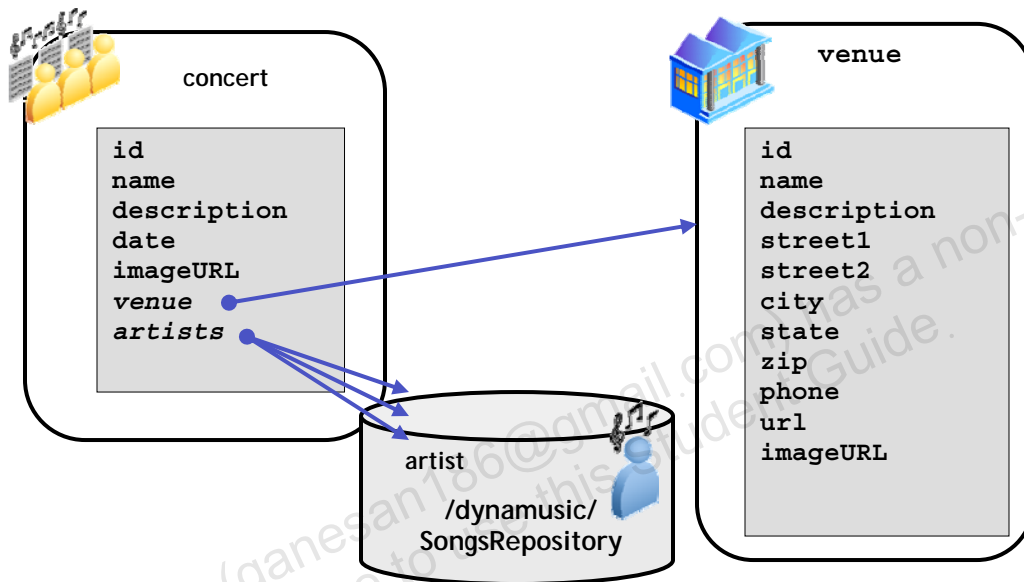
Practice 5-1: Configuring and Using SQL Repositories

Practice Objective

After completing this practice, you should be able to create and configure a SQL repository.

Problem Statement

In this practice, you will create an Events Repository that will contain venue and concert information. The SQL database tables associated with venues and concerts have already been created in the MySQL database. These tables are currently empty. You will populate them with repository-item data as part of this practice.



For help with the syntax of the repository definition file, refer to the lecture notes or the “SQL Repository Reference” chapter in the *Repository Guide*.

The SQL definitions of the Events (venues and concerts) tables follow:

```
CREATE TABLE dynamusic_venue (  
    id                VARCHAR(32) not null,  
    name              VARCHAR(100)   null,  
    description        LONG   VARCHAR null,  
    street1            VARCHAR(100)   null,  
    street2            VARCHAR(100)   null,  
    state              VARCHAR(32) null,  
    city               VARCHAR(50) null,  
    zip                VARCHAR(10) null,  
    phone              VARCHAR(20) null,  
    image              VARCHAR(100) null,  
    url                VARCHAR(100) null,  
    primary key(id)  
);
```

```

CREATE TABLE dynamusic_concert (
    id          VARCHAR(32)      not null,
    name        VARCHAR(100)     null,
    description  LONG VARCHAR    null,
    venue       VARCHAR(32)      null
    references dynamusic_venue(id),
    image       VARCHAR(100)     null,
    event_date  TIMESTAMP        null,
    primary key(id)
);

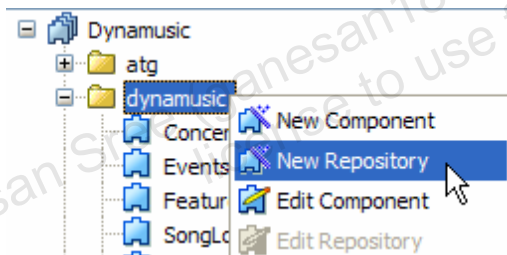
CREATE TABLE dynamusic_concert_artists (
    concert_id VARCHAR(32)      not null
    references dynamusic_concert(id),
    artist_id  VARCHAR(32)     not null
    references dynamusic_artist(id),
    primary key(concert_id, artist_id)
);

```

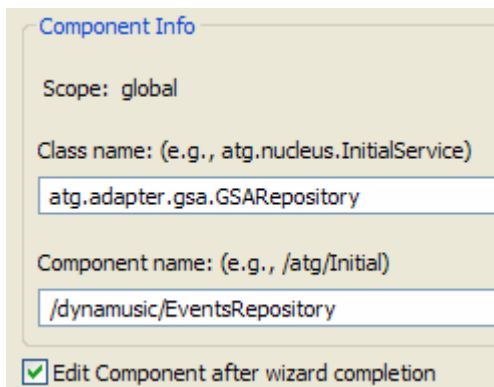
Guided Instruction

Define the Events Repository Component

1. In Eclipse, view the ATG Component Browser (select **Window** → **Show View** → **Other** → **ATG** → **ATG Component Browser** if not currently visible). Browse to the **dynamusic** component folder, right-click, and select **New Repository** as shown below. Be careful not to select New Component!



2. For the repository, accept the default base class, and set the component name to `/dynamusic/EventsRepository`. Select the option to edit the component after wizard completion and click **Finish**.



3. The component properties file will be created. Review the properties set as default for you. In particular, note the value of `definitionFiles`. It defaults to an XML file with the same name as the repository (although you can specify any value). Close the component properties window when you are done.

Define the Events Repository Item Types

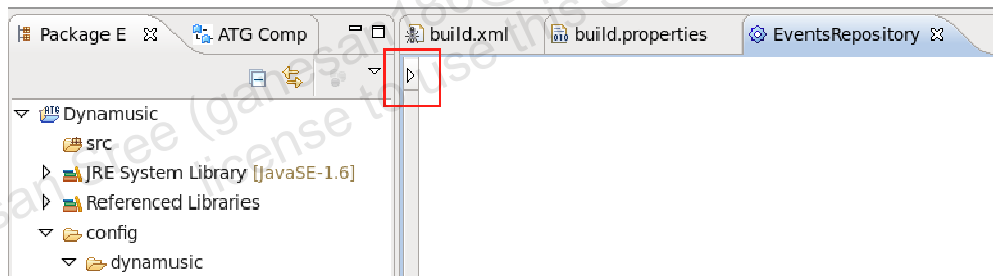
The ATG Eclipse plug-in provides a visual editor that will help you define your repository data mappings. The instructions below assume you will use the visual editor. If you are comfortable with XML, you may choose to edit the definition file (`atgdir/Dynamusic/config/dynamusic/EventsRepository.xml`) directly instead using the starting data from the `atgroot/Training/Foundations/src/chapter05/EventsRepository.xml` file.

Editing XML: Eclipse includes a DTD-aware XML editor (part of the Web Tools set of plug-ins). The XML editor includes features such as design, source, and outline views, and Ctrl + space for contextually valid tags and attributes. For more information on using the Eclipse Web Tools XML editor, see Help → Help Contents → Web Application Development User Guide → Editing Markup Language Files.

1. Start the Repository Definition Editor by right-clicking the `EventsRepository` component in the ATG Component Browser and select **Edit Repository**.

Note that an alternative way to open the Repository Definition Editor is to navigate in Eclipse to the specific XML file (`config/dynamusic/EventsRepository.xml`), right click the file, and choose **Open With → Repository Editor**.

2. Open the list of Repository objects in the object palette as shown below:

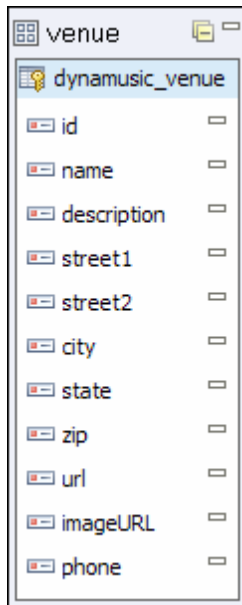


and then drag an **Item Descriptor** object and place it in the Repository Definition Editor window. Enter item type name: `venue`.



3. Drag a **table** object from the repository editor palette to your new item descriptor box and change its name to `dynamusic_venue`.

4. All of the `venue` item type's properties are stored in the same table. Drag a **property** object into the `dynamusic_venue` table for each property in the item type and set the name of the property:




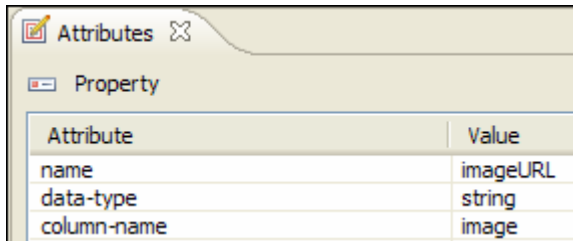
5. If the Attributes panel isn't visible, select **Window → Show View → Other → ATG → Attributes (or simply Window → Show View → Attributes** if you are using the ATG Perspective) from the Eclipse menu bar. You can click the item, table, or property for each element you added to see a view of the Attributes, such as those for the `dynamusic_venue` table as shown below:

A screenshot of the 'Attributes' panel in Eclipse. The panel shows a table with two columns: 'Attribute' and 'Value'. The table contains three rows of data.

Attribute	Value
name	dynamusic_venue
type	primary
id-column-names	id

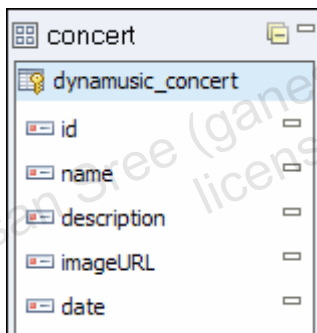
As you click each item property you added, the attributes panel shows the attributes defined for each property. Click each item property and note its attributes.

- By default, the repository editor in Eclipse will map all properties to database columns of the same name. In this example, most properties will follow this pattern, but not the `imageUrl` property, which you need to map to the `image` column. Select the `imageUrl` property and in the attributes panel, click the **Add**  button. From the pop-up list of attributes, choose `column-name` and click **OK**. The new attribute should appear in the list. Give it the value `image`.



Attribute	Value
name	imageUrl
data-type	string
column-name	image

- By default, the display-properties for repository items are the items' IDs. Click the `venue` item descriptor to display its attributes, and add an attribute definition for `display-property` to set it to `name`.
- Use the object palette to create a second item type called `concert`. You can place it to the right of the `venue` item type you just finished creating.
- The `concert` item's simple properties are in a table called `dynamusic_concert`; add that table object to the `concert` item type.
- Add the following properties to the `dynamusic_concert` table:



concert	
dynamusic_concert	
id	
name	
description	
imageUrl	
date	

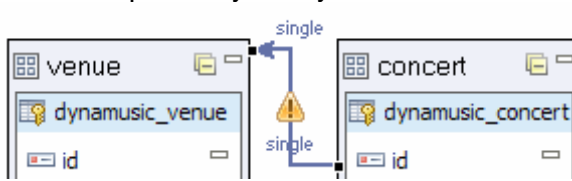
- Set the `column-name` attribute of `imageUrl` to `image`. Set the `column-name` attribute of `date` to be `event_date`.
- By default, all properties are assumed to be of type `string`. Change the `data-type` attribute for the `date` property to `timestamp`.

Attribute	Value
name	date
data-type	timestamp
column-name	event_date

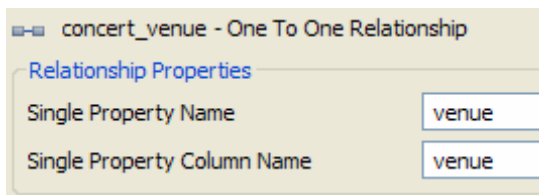
- Set the `display-property` attribute for the `concert` item type to `name`.

Define a One-to-One Relationship

1. Under the Relationships folder in the Palette, click One to One. Then first click the **concert** item type, then the **venue** item type, to establish a relationship between these two item types. The relationship will be displayed with a yellow triangle indicating that the relationship is not yet fully defined.



2. Right-click the triangle and select **Edit Relationship**. In the relationship editor, change the Single Property Name to `venue` and click **OK**.



3. Save the repository definition by selecting **File → Save**.
4. Note that the XML file generated by the Repository Definition Editor can be easily found on the project navigation panel, with the following path:
`Dynamusic/config/dynamusic/EventsRepository.xml`.
5. Next you need to check your work done so far using the provided `TestDynamusic` module with `startSQLRepository`. The `TestDynamusic` module is very simple; its `MANIFEST.MF` file sets a dependency on `Dynamusic`, and its `config` folder overrides the `JTDataSource` component used by `Dynamusic` repositories. The override transforms it into an ATG-managed, rather than application-server-managed, data source. It continues to refer to the same database you have been using all along.

Open a terminal window, change directory to `atgdir/home`, and enter the following (all on one line):

```
bin/startSQLRepository -m TestDynamusic -repository
/dynamusic/EventsRepository
```

Resolve any errors or warnings before proceeding.

Add a One-to-Many Relationship Between Concert and Artist

1. For the next steps, it is a good idea to edit the XML directly to broaden your experience with repository definition files. Close the editing windows currently open in Eclipse, then click the **Navigator** tab and right-click `Dynamusic/config/dynamusic/EventsRepository.xml`. Choose **Open With > XML Editor**. Click the **Source** tab at the bottom of the editing screen.

2. Add a second table tag, referring to a table called `dynamusic_concert_artists`, to the `concert` item descriptor with additional attributes as follows:

type	multi
id-column-names	concert_id

3. Add an `artists` property to this new table with additional attributes as follows:


column-name	artist_id
data-type	set
component-item-type	artist
repository	/dynamusic/SongsRepository

4. Save your repository definition. (You may get a warning about an invalid foreign repository reference. It can be safely disregarded.)
5. Test your repository again as you did above, by running `startSQLRepository` with the `TestDynamusic` module. Verify that no errors are reported.

Register the Repository

1. To both manage the Events Repository via the ACC *and* have it started automatically whenever ATG start up, open the `/atg/registry/ContentRepositories` component for editing via the Component Browser. Add your new repository to the `initialRepositories` property. (You can edit a list type property by right-clicking the property name and selecting Edit.)

Collection Values
/atg/userprofiling/ProfileAdapterRepository
/atg/content/media/MediaRepository
/atg/userprofiling/PersonalizationRepository
/dynamusic/SongsRepository
/dynamusic/EventsRepository

2. Shut down the ACC, run the `deploy Ant` task, and restart the ACC.
3. Use the ACC to add several venue and concert items. Select **Content** → **EventsRepository**, and then click the **New Item**  **New Item...** button. Begin by creating items of type `venue`, and then create a few `concerts` for each venue.

Import Predefined Items into the Events Repository

To save you some time, several predefined `venue` and `concert` items are provided for you in repository `<add-item>` XML format. All you have to do is run `startSQLRepository` to import the predefined items.

1. Copy `events-data.xml` from the `atgroot\Training\Foundations\src\chapter05\` folder to the `atgdir\Dynamusic\config\dynamusic\` folder.

2. From a cmd prompt window, change directory to *atgdir/home*, and enter the following command:

```
bin/startSQLRepository -m TestDynamusic -import  
../Dynamusic/config/dynamusic/events-data.xml -repository  
/dynamusic/EventsRepository
```

3. Use the ACC to list venue and concert items. Verify that the predefined items have been added to the repository.

Display a List of Venues

1. You have been provided with “drill-down” pages for the Events Repository, which use precreated components *VenueLookupDroplet* and *ConcertLookupDroplet*. If your repository is configured correctly as described, these pages should work for you. (*venues.jsp*, *venueDetails.jsp*, and *concertDetails.jsp*.) Load the **Venues** page, select a venue and verify your changes.
2. **Optional:** As time allows, create “drill-down” pages (and required lookup droplets) for Events Repository on your, rather than using the provided ones. The steps will be very similar to those required for setting up the *artists.jsp*, *artistDetails.jsp*, and *albumDetails.jsp* pages in the previous practice.

Practice 5-2: Creating Item Subtypes

Practice Objective

After completing this practice, you should be able to create repository item subtypes.

Problem Statement

Dynamusic wants to be able to distinguish between general concerts and concerts that are fundraisers for a specific organization. For fundraisers, concert items should include the name of the organization for which funds are being raised.

Define a new item type called `concert-fundraiser`, which is a subtype of `concert`, and add a property called `organization`.

The SQL database table columns associated with this extension have already been defined in the `dynamusic_concert` and `dynamusic_concert_fundraiser` tables:

```
CREATE TABLE dynamusic_concert (
    ...
    event_type    VARCHAR(20)    null,          for sub-type property
    ...
);
CREATE TABLE dynamusic_concert_fundraiser (
    concert_id    VARCHAR(32)    not null
                references dynamusic_concert(id),
    fund_org      VARCHAR(50)    null,          for organization property (stores its name)
                primary key(concert_id)
);
```

Guided Instruction

Implement a `concert-fundraiser` Subtype for `concert`

There are two steps to implementing a subtype. First, modify the base item type to define a subtype property (in this case, the `concert` item). Next, define the subtype item with its additional properties (`concert-fundraiser`).

1. Edit the `EventsRepository` definition file directly in the Eclipse XML editor (`atgdir/Dynamusic/config/dynamusic/EventsRepository.xml`), or alternatively, use the visual Repository Editor.
2. Add a `concertType` property to the `dynamusic_concert` table in the `concert` item, mapped to the `event_type` column. Make it an enumerated property and create two options: `concert` and `fundraiser`. Set the `code` attributes for these options to any integers you like and the `value` attributes to `concert` and `fundraiser`, respectively. Note that setting the value attribute for an option tag must be edited in the xml.
3. Add an `attribute` tag to the `concertType` property and name it `useCodeForValue`. Set the value of the attribute to `false`.
4. Add the `sub-type-property` attribute to the `concert` item and set its value to `concertType`.

5. Create a new item descriptor for the subtype named `concert-fundraiser`. Set the `super-type` and `sub-type-value` attributes appropriately.
6. Add the `dynamusic_concert_fundraiser` table (as an auxiliary table) and set its `id-column-name` value appropriately. Add the single property this item subtype requires: `organization` (type `string`, mapped to column `fund_org`).
7. Test your work using `startSQLRepository`.
8. Shut down the ACC, run the `deploy Ant` task, and restart the ACC.
9. Use the admin interface in the browser to restart the ACC. In the ACC, create a new concert item, selecting `concert-fundraiser` as the item type. Verify that the `organization` property is displayed in the component configuration box. Select a new or existing venue for the concert.

New "concert-fundraiser" Values	
artists	[Frida Jetsen, Autumn Winters]
concertType	fundraiser
date	May 4, 2011 1:15:40 PM EDT
description	The two best female pop stars of th...
id	270725
imageURL	/images/PopForFish.jpg
name	Pop Girls for Fish 2012
organization	Save the Parrotfish
venue	1870 Town Hall

Optional: Display Subtype-Specific Information in a Page

1. Edit `concertDetails.jsp` and use the `Switch` servlet bean to display a line that lists the fundraising organization if the concert type is `fundraiser`. The input to the `Switch` droplet should be the concert's `concertType` property. The `oparams` of the `Switch` bean must match possible values for the switch input: in this case, a single `oparam` called `fundraiser`. Run the `update_files Ant` task.
2. Navigate to the event detail page and verify that the fundraiser organization is displayed for your new `concert-fundraiser`.

Practices for Lesson 6: Using Repository Form Handlers

Chapter 6

Catching Up

If you were unable to complete the practices from the lesson titled “Configuring SQL Repositories,” perform the following steps:

1. Complete the “Catching Up” instructions from practice 5, if you did not complete those practices.
2. Copy the `ContentRepositories.properties` file in `coursedir/solutions/chapter05` to `atgdir/Dynamusic/config/atg/registry`. You will need to create the `atg` and `registry` folders.
3. Copy the remaining properties files and the XML file in `coursedir/solutions/chapter05` to `atgdir/Dynamusic/config/dynamusic`.
4. Copy the `events-data.xml` file in `coursedir/src/chapter05` to `atgdir/Dynamusic/config/dynamusic`.
5. Restart ATG.
6. From a cmd prompt window, change directory to `atgdir/home`, and enter the following command:

```
bin/startSQLRepository -m TestDynamusic -import
../Dynamusic/config/dynamusic/events-data.xml
-repository /dynamusic/EventsRepository
```

7. In the Eclipse Navigator, refresh your Dynamusic project by right-clicking the Dynamusic folder and selecting **Refresh**.

Practice 6-1: Using Repository Form Handlers

Practice Objective

After completing this practice, you will be able to use Repository Form Handlers to add, delete, and modify repository items. In addition, you will be able to use the Search Form Handler to search for repository items meeting specific criteria.

Problem Statement

In this practice you will be asked to create several JSPs that contain forms. As with previous practices, you are provided with formatted starter JSP files. In these starter files, you will find nicely formatted static form templates that use tables to control how the form fields will look on the page. As always, you may choose instead to use `dynamusic-template.jsp` as a “blank” starter page if you find working with the formatted HTML difficult.


Guided Instruction

Implement a Form to Add a New Artist

1. Create a new **request scoped** form handler component called `ArtistFormHandler` based on the `atg.repository.servlet.RepositoryFormHandler` class. Set the component's properties as follows:
 - `itemDescriptorName`: `artist`
 - `repository` links to `/dynamusic/SongsRepository`
 - `requireIdOnCreate`: `false`
 - `clearValueOnSet`: `false`
2. Open the `coursedir/examples/chapter06/newVenue.jsp` file for review. You won't make any edits in this file but you may find it useful as an example of the types of tags needed in the pages that you will be editing in the following steps. You may find it helpful to copy and paste from `newVenue.jsp` as you proceed below.
3. Open the provided `newArtist.jsp` starter file for editing (or create a new one using `dynamusic-template.jsp`).
4. Import the `/dynamusic/ArtistFormHandler` component (bean).
5. Locate the `<form...> ... </form>` tags and change them to `<dsp:form> ... </dsp:form>` tags. Change the “action” of the form tag to redirect back to the *current page*.
6. Your form should collect the `name`, `photoURL`, and `description` properties of the new artist. If you are using the mock-up starter file, locate the static fields in the form and replace them with DSP form input tags, which link input fields to associated item properties in the form handler. Remember to connect each form input field to the associated property using the `ArtistFormHandler.value.myProperty` syntax. Review the lecture notes or the DSP tag library appendix of the *Page Developer's Guide* for details on DSP form tag syntax. Make the `name` property required.
7. Link the Submit button (`<dsp:input type="submit" ...>`) to the appropriate submission handler method (`ArtistFormHandler.create`).
8. Add a hidden input field to set the `createSuccessURL` property to `artists.jsp`.

9. At the beginning of the form, handle errors by looping through the form handler's `formExceptions` property using the `ErrorMessageForEach` servlet bean. For each error, display the message output parameter.
10. Edit `artists.jsp` and link the text "Add Artist" (under the sidebar) to `newArtist.jsp`. Run the `deploy Ant` task.
11. Load the **Artists** page and click the "Add Artist" link. Add an artist, and verify your changes. (Note that `photoURL` is limited to 32 characters because of the database column size.)
12. Try to add an artist with no name entered on the form. Verify that the form is redisplayed with an error indicating that the "name" field is required.

Implement a Form to Edit or Delete an Artist

1. Edit `editArtist.jsp`. Import the `/dynamusic/ArtistFormHandler` component bean.
2. Create a form to allow the user to modify the name, description, or `photoURL` of the artist whose ID is passed in by the `artistId` parameter. Make the name field required. If you are using the provided mock-up, locate the static form and form fields, and replace each tag with the corresponding DSP form tag. Connect each input field with the bean property (similar to the previous practice.) Be sure to set the "action" of the form to the *current JSP*.
3. As with `newArtist.jsp`, handle errors at the top of the form.
4. Link the "Save Changes" button to the `ArtistFormHandler.update` method, and the "Delete this Artist" button to the `ArtistFormHandler.delete` method.
5. Set the redirect properties `updateSuccessURL` and `deleteSuccessURL` to `artistDetails.jsp` and `artists.jsp`, respectively. See lecture notes for help on passing the parameter to your `updateSuccessURL` page.
6. The form should modify the artist item defined by an input parameter called `artistId`. In order for the `ArtistFormHandler` to know which artist you are modifying, you need to set the `repositoryId` property of the form handler. Just above your form, add a `<dsp:setvalue>` tag to set the `repositoryId` property of the form handler to the `artistId` parameter passed in from the calling page.
7. Make sure you use that parameter to set `ArtistFormHandler.repositoryId` via a hidden form input field as well. You must give this field a name, specifically `artistId`, in order for the form error handling to work when the page is redisplayed. (See the lecture notes for details.)
8. Edit `artistDetails.jsp` and link the text "Edit Artist Info" (under the sidebar) to `editArtist.jsp`, making sure to pass the `artistId` as a parameter. Run the `update_files Ant` task.
9. Load an artist detail page and click the **Edit Artist Info** link. Modify the artist details and verify your changes. (You can either revisit the artist page or use the ACC to verify changes; be sure to refresh your ACC  to show the latest data.)
10. Edit the details for an artist, leaving the name field blank, and verify the form error handling.
11. Test the deletion function. It might fail due to foreign key violations if there are albums defined for that artist. For now, try creating a new artist (using your previously created form or the ACC) with no albums and then deleting that artist to make sure the form works.

Optional: Implement a Song Search Page

1. Create a session-scoped component in the `/dynamusic` folder called `SongSearchFormHandler` based on the `atg.repository.servlet.SearchFormHandler` class. Set the `doKeywordSearch` property to `true`, the `itemTypes` property to `song`, link the `repositories` property to `/dynamusic/SongsRepository` and the `keywordSearchPropertyName` property to `song.title`.
2. Modify `search.jsp` and add a form with a single text box that allows users to search for songs by keyword. Connect the text box to the `keywordInput` property of the form handler. Add a hidden input field to set the `successURL` property to `searchResults.jsp` (which you will create in the next step), and add a Submit button to invoke the `search` method of the form handler.
3. If you are using the ACC to edit your pages, please note that the ACC does not provide a wizard to implement a search form. You can still use the wizard to set up form elements such as automatic error processing.
4. Modify `searchResults.jsp` to display the results of the user's search. Insert a `ForEach` servlet bean to loop through the results (`SongSearchFormHandler.searchResults`). Display the name of the song, as well as the artist, as links to the respective details pages (`song.jsp` and `artistDetails.jsp`), making sure to pass a parameter containing the `repositoryId` of the song and the artist, respectively. Run the `update_files` Ant task.
5. Load the home page and select **Search**. Enter a substring that matches the name or description of multiple songs in the Song Repository ("Love" is always a good choice!) and verify your changes.

Ganesan Sree (ganesan186@gmail.com) has a non-transferable license to use this Student Guide.

Practices for Lesson 7: Extending the Profile Repository

Chapter 7

Catching Up

If you were unable to complete the practices for the lesson titled “Using Repository Form Handlers,” perform the following steps:

1. Complete the “Catching Up” section from Practice 6, if you did not complete those practices.
2. Copy the properties files in *coursedir/solutions/chapter06* to *atgdir/Dynamusic/config/dynamusic*.
3. Copy the JSP files in *coursedir/solutions/chapter06* to *atgdir/Dynamusic/j2ee-apps/dynamusic-j2ee/dynamusic-web.war*.
4. In the Eclipse Navigator, refresh your Dynamusic project by right-clicking the Dynamusic folder and selecting **Refresh**.

Practice 7-1: Extending the Profile and Using the Profile Form Handler

Practice Objective

After completing this practice, you should be able to add or modify properties in the user profile definition.

Problem Statement

In this practice, you will extend the profile definition to include custom properties. The SQL database tables associated with these extensions have already been created in the database. These tables are currently empty. You will populate them with user data when you verify your work.

The SQL definitions of the Dynamusic profile extension tables are as follows:

```
CREATE TABLE dynamusic_user (
    user_id          VARCHAR(32)      not null
        references dps_user(id),
    info             LONG VARCHAR     null,
    share_profile    NUMERIC(1)       null,
        CHECK (share_profile in (0, 1)),
    primary key(user_id)
);

CREATE TABLE dynamusic_viewedartists (
    user_id          VARCHAR(32)      not null
        references dps_user(id),
    artist_id        VARCHAR(32)      not null
        references dynamusic_artist(id),
    primary key(user_id, artist_id)
);

CREATE TABLE dynamusic_prefgenres (
    id               VARCHAR(32)      not null
        references dps_user(id),
    genre            VARCHAR(32)      not null,
    primary key(id, genre)
);
```

Guided Instruction

Extend the Profile with Custom Properties

Although you can use the Eclipse ATG plug-in Repository Editor as you did in the lesson titled “Configuring SQL Repositories,” because the Profile Repository is fairly complex to begin with, you will probably find it simpler to edit the XML file manually using the Eclipse XML editor. A starter file has been provided to you for this purpose.

Refer to the slides and notes for the lesson titled “Configuring SQL Repositories” for a reminder on how to define item types in a repository definition file. Also remember that in the Eclipse XML editor, pressing Ctrl + Space will bring up a context-sensitive list of valid attributes and child tags.

1. Import `userProfile.xml` from `atgroot/Training/Foundations/src/chapter07` to `Dynamusic/config/atg/userprofiling`. In Eclipse, you will have to create the `userprofiling` subfolder. When importing, add the xml file to the “Into Folder”.
2. Add three new table tags (containing the four new properties) to the description of the user item. (Note that the tables do not all have the same name for the ID column.)

Table: `dynamusic_user`

Type: `auxiliary`

Id Column: `user_id`

Property Name	Type	Column Name
<code>info</code>	<code>string</code>	<code>info</code>
<code>shareProfile</code>	<code>boolean</code>	<code>share_profile</code>

Table: `dynamusic_prefgenres`

Type: `multi`

Id Column: `id`

Property Name	Type	Component Type	Column Name
<code>prefGenres</code>	<code>set</code>	<code>string</code>	<code>genre</code>

Table: `dynamusic_viewedartists`

Type: `multi`

Id Column: `user_id`

Property Name	Type	Component Type	Repository	Column Name
<code>viewedArtists</code>	<code>set</code>	<code>artist (item type)</code>	<code>/dynamusic/ SongsRepository</code>	<code>artist_id</code>

The `info` property will be used to display information you wish to share with other users who can see your profile. Permission to share this data will be controlled by the Boolean `shareProfile` property. Realistically, you would probably have a “share” type Boolean property for each different attribute you might want to show, but for simplicity in this practice, you will be implementing one such property for the whole profile. The `prefGenres` property will keep track of a set of the user’s favorite types of music, and the `viewedArtists` property will keep track of the artists (set of item pointers) viewed by the user. (The `viewedArtists` property would typically be updated by a scenario, which is covered in the *ATG Personalization and Scenario Development* course; for now, you can test it by populating it manually in the ACC.)

Include a `category` attribute for your custom properties, grouping them under `dynamusic`.

3. Test your work using `startSQLRepository`. Open a cmd prompt window, change directory to `<atgdir>/home` and enter the following command:

```
bin/startSQLRepository -m TestDynamusic -repository
/atg/userprofiling/ProfileAdapterRepository
```

 Resolve any errors before proceeding.
4. Shut down the ACC, run the `deploy` Ant task, and restart the ACC.
5. In the ACC, select **People and Organization** → **Users** and select a user. Verify and inspect the new profile properties via the ACC by executing the following tasks:
 - Verify that the **dynamusic** category is displayed and includes the `info`, `prefGenres`, `shareProfile`, and `viewedArtists` properties.
 - Verify that you can add artists to a user’s `viewedArtist` property.
 - Examine the ATG logging window to see if any errors are displayed.

Practice 7-2: Using the Profile Form Handler

Practice Objective

After completing this practice, you will be able to use the Profile Form Handler to create a JSP that allows users to register. The same pattern is used in JSPs that allow users to update their profile, log in, and log out.

Problem Statement

In this practice, you will add a registration page for new users.

Guided Instruction

Implement a registration page

1. Edit `newProfile.jsp`. A formatted HTML form has been provided. Note that the `<dsp:page>` tag has already been added to the page, and that the `ProfileFormHandler` component has been imported into the page for you. Modify the form to be a DSP form. Attach the input fields to the following properties of the `/atg/userprofiling/ProfileFormHandler.value` dynamic bean property.
 - `login` (required, text)
 - `password` (required, password)
 - `firstName` (text)
 - `lastName` (text)
 - `homeAddress.state` (text)
 - `prefGenres` (check boxes)
 - `shareProfile` (Boolean/option button)
 - `info` (text area)
2. Set the **Cancel** and **Register** buttons to call the appropriate methods, and use a hidden input to set `createSuccessURL` and `cancelURL` properties to `home.jsp`.
3. Include a password confirmation field by including an extra text input field for the password confirmation, attaching the input value to `ProfileFormHandler.value.confirmPassword`. Set the `ProfileFormHandler.confirmPassword` property to true via a hidden input tag. Use the input type `password` for the input field to hide the password typing.
4. Save your page.
5. Edit `home.jsp` and edit the `Switch` servlet bean around the welcome message to check the value of `Profile.transient`. If true, indicating that the user is not logged in, display a link to `newProfile.jsp` to allow the user to register. Otherwise (the user is logged in), display the old welcome message.

6. Modify `/common/sidebar.jsp` to also check the value of `Profile.transient`. If `true`, include a link to `login.jsp`. If `false`, include a link to `logout.jsp`, as well as a link to the `updateProfile.jsp`. Remember that to use DSP tags, you must declare the tag library and enclose the page within the `<dsp:page>` tag.
7. Run the `update_files` Ant task.
8. Load the Dynamusic home page and register, log out, and log back in to test your work.

Ganesan Sree (ganesan186@gmail.com) has a non-transferable license to use this Student Guide.

Ganesan Sree (ganesan186@gmail.com) has a non-transferable license to use this Student Guide.