

Extending the ATG Platform Rel 10.1.2

Activity Guide

D81525GC10
Edition 1.0
May 2013
D82004

ORACLE

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.

Author

Karin Layher

Technical Contributors and Reviewers

Jay Cross, Tom Hardy

This book was published using: **Oracle** Tutor

Table of Contents

Practices for Lesson 1: Creating Custom Components	1-1
Practices for Lesson 1: Overview.....	1-2
Practice 1-1: Setting Up for Java Development	1-3
Practice 1-2: Creating a Custom Component.....	1-9
Practices for Lesson 2: The Repository API.....	2-1
Practices for Lesson 2: Overview.....	2-2
Practice 2-1: Catching Up	2-3
Practice 2-2: Using the Repository API	2-4
Practices for Lesson 3: Custom Form Handlers	3-1
Practices for Lesson 3: Overview.....	3-2
Practice 3-1: Catching Up	3-3
Practice 3-2: Extending the Repository Form Handler	3-4
Practice 3-3: Creating a Custom Form Handler	3-6
Practices for Lesson 4: Managing Transactions.....	4-1
Practices for Lesson 4: Overview.....	4-2
Practice 4-1: Catching Up	4-3
Practice 4-2: Controlling Transactions	4-4
Practices for Lesson 5: Complex Repository Configuration.....	5-1
Practices for Lesson 5: Overview.....	5-2
Practice 5-1: Catching Up	5-3
Practice 5-2: Configuring a Many-to-Many Relationship	5-4
Practices for Lesson 6: Advanced SQL Repository Configuration	6-1
Practices for Lesson 6: Overview.....	6-2
Practice 6-1: Catching Up	6-3
Practice 6-2: Creating a Custom Property.....	6-4
Practices for Lesson 7: Derived Properties	7-1
Practices for Lesson 7: Overview.....	7-2
Practice 7-1: Catching Up	7-3
Practice 7-2: Creating a Derived Property.....	7-4
Practice 7-3: Making a Derived Property Overrideable	7-5
Practices for Lesson 8: Repository Caching	8-1
Practices for Lesson 8: Overview.....	8-2
Practice 8-1: Catching Up	8-3
Practice 8-2: Preloading the SQL Repository Cache	8-4
Practices for Lesson 9: Custom Droplets	9-1
Practices for Lesson 9: Overview.....	9-2
Practice 9-1: Creating a Custom Servlet Bean.....	9-3

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

Practices for Lesson 1: Creating Custom Components

Chapter 1

Practices for Lesson 1: Overview

Practices Overview

In these practices, you set up an Eclipse project for Java development on an ATG module, and then you create a custom component.

Practice 1-1: Setting Up for Java Development

Overview

In this practice, you set up for Java development by doing the following:

- Declaring the `DynamusicEAP` module's classpath
- Creating an Eclipse project for the module and setting up its Java build environment
- Enabling the Eclipse Java debugger to work with WebLogic

Assumptions

It is recommended that you use Eclipse (with the ATG plug-in) for editing all XML and Java files throughout the practices. The ATG Eclipse plug-in has been installed as part of the setup of this course.

The main module for this class is `DynamusicEAP`, which has already been created and installed for your convenience.

The following files and folders are used in these practices:

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

Tasks

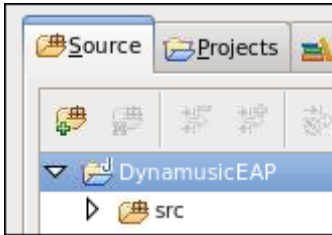
Edit the Module's Manifest File

1. Edit the `atgdir/DynamusicEAP/META-INF/MANIFEST.MF` file. Add a line setting the `ATG-Class-Path` to `classes/`.

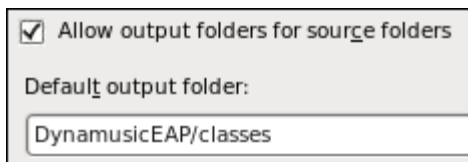
Create a New Eclipse Project for the Existing Module

2. Make sure you have completed step 1 before you proceed. On your desktop, locate the **EAP_shortcuts** folder, open it, and double-click **startEclipse**. If you are prompted for a workspace location, accept the default provided.
3. If necessary, switch to the ATG perspective by selecting **Window > Open Perspective > Other**, selecting **ATG**. Click **OK**.
4. Select **File > New > Existing ATG Module**. In the resulting dialog box, type `DynamusicEAP` into the Project field. Click **Next**.

5. The **Java Settings** dialog box opens. The **Source** tab displays the default source folder with a special folder icon (DynamusicEAP/src). This is the folder that you use in class, so leave it unchanged.



Select the **Allow Output Folders for Source Folders** option so that compiled Java code is stored in a separate output folder instead of the source folder. The default output folder should be set to `DynamusicEAP/classes`. Click **Finish**.

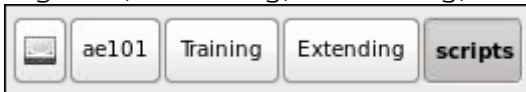
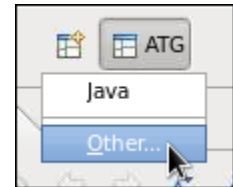


6. The **DynamusicEAP** project should now appear on the Package Explorer tab on the left. Open the project to see the folders that appear on the Package Explorer tab.

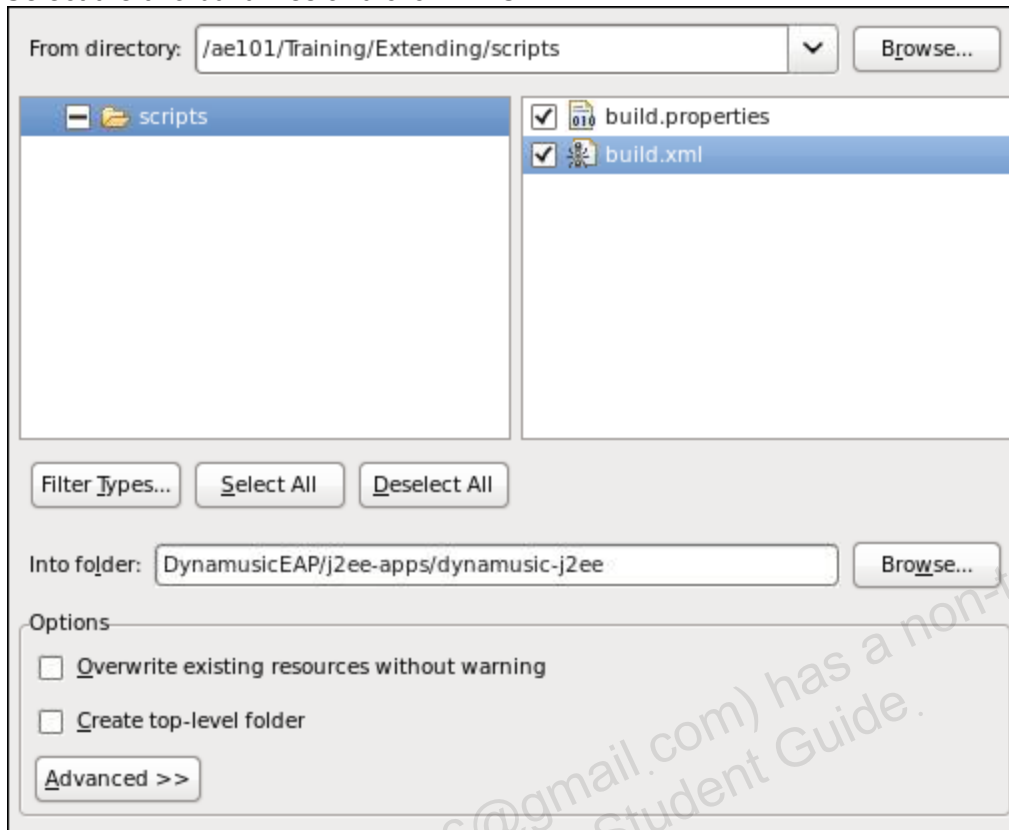
Import the Dynamusic Build Files

For this course, an ANT build script is used to update Web files on a running instance of WebLogic Server. Additionally, the script can undeploy, assemble, and deploy the DynamusicEAP application. This section covers how to import the build files you need.

7. In Eclipse, open the Java EE perspective by choosing **Window > Open Perspective > Other** and selecting **Java EE**. Alternatively, click the Open Perspective button in the top-right corner of Eclipse and select **Other**. Select **Java EE** and click **OK**. Your project now appears on the Project Navigator tab. Open the `DynamusicEAP` project and locate the `j2ee-apps/dynamusic-j2ee/` folder. Right-click the folder and select **Import**.
8. Under the **General** folder, select **File System** as the import source and click **Next**.
9. For the source directory, browse to and select `atgroot/Training/Extending/scripts/`. Click **OK**.



10. Select the two build files and click **Finish**.



Start WebLogic Server and the ATG Server

11. Start WebLogic Server. From the `EAP_Shortcuts` folder on your desktop, double-click **startWebLogicServer**. When prompted, choose **Run in Terminal**. You may want to set the title for the terminal window to something like “WebLogic Server” (**Terminal > Set Title**). When you see the message stating that the server called “AdminServer” is started in running mode, the startup has completed:

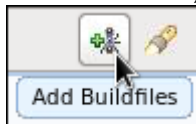
```
<Apr 18, 2013 7:43:48 AM UTC> <Notice> <WebLogicServer> <BEA-000331> <Started WebLogic Admin Server "AdminServer" for domain "atg_education" running in Development Mode>
<Apr 18, 2013 7:43:48 AM UTC> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to RUNNING>
<Apr 18, 2013 7:43:48 AM UTC> <Notice> <WebLogicServer> <BEA-000360> <Server started in RUNNING mode>
```

12. Start the ATG server similarly to the previous step by double-clicking **startATGServer**. You may want to set the title for the terminal window to something like “ATG Server.” This server takes longer to start than WebLogic Server. When you see the message stating that the managed server “eapprod” is started in running mode, the startup has completed:

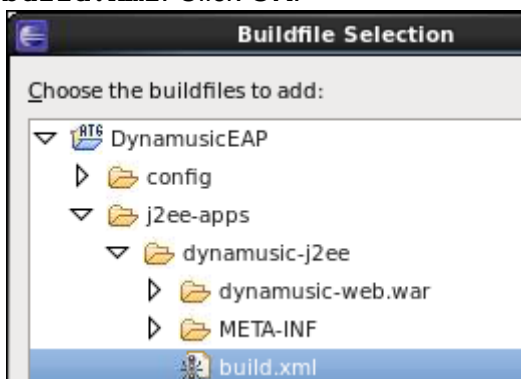
```
<Apr 18, 2013 7:49:12 AM UTC> <Notice> <WebLogicServer> <BEA-000332> <Started WebLogic Managed Server "eapprod" for domain "atg_education" running in Development Mode>
<Apr 18, 2013 7:49:14 AM UTC> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to RUNNING>
<Apr 18, 2013 7:49:14 AM UTC> <Notice> <WebLogicServer> <BEA-000360> <Server started in RUNNING mode>
```

Deploy Your Application

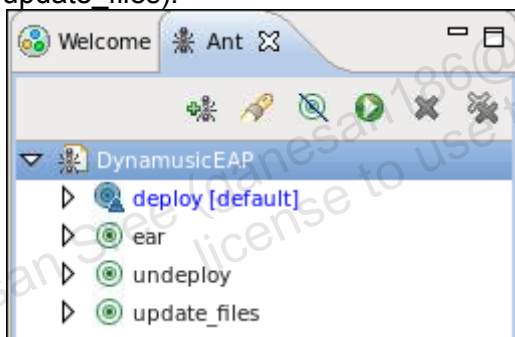
13. In Eclipse, make sure that the Ant tab is viewable by going to **Window > Show View > Other**. Open the Ant folder, select **Ant**, and click **OK**. This step adds the Ant tab to Eclipse.
14. On the Ant tab, click the Add Buildfiles icon.



15. Navigate to the `/j2ee-apps/dynamusic-j2ee` folder in your project and select `build.xml`. Click **OK**.



16. Expand DynamusicEAP. You should see the four tasks (deploy, ear, undeploy, and update_files):



17. Double-click the **deploy** task. This runs three of the tasks in the following order: undeploy, ear, and deploy. The undeploy task removes the DynamusicEAP application from your ATG server under the WebLogic file system. The ear task assembles all the files and libraries needed to create the ear file that can be deployed to the server. The deploy task deploys the ear file to the server. You can follow the progress on the Console tab; you know it has completed when you see something similar to the following messages:

```
[wldeploy] Task 1 initiated: [Deployer:149026]deploy application DynamusicEAP on eapprod.  
[wldeploy] Task 1 completed: [Deployer:149026]deploy application DynamusicEAP on eapprod.  
[wldeploy] Target state: deploy completed on Server eapprod  
[wldeploy] Target Assignments:  
[wldeploy] + DynamusicEAP eapprod  
[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: 44 seconds
```

Note: The deploy task causes the ATG server to restart. The messages in the terminal

window often stop after the “Nucleus running” message. This is fine. You can proceed with the practice after you see that message.

Test Your Application

18. Confirm that the application is running correctly by visiting the application’s home page at <http://localhost:7103/dynamusic>, and browse the artists and venues. (Bookmark this page because you visit it frequently throughout the practices.) Note that there is also a shortcut in your desktop folder (EAP_Shortcuts) called Open DynamusicEAP.
19. Go 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. Note that there is also a shortcut in your desktop folder (EAP_Shortcuts) called Open Dynamo Admin.

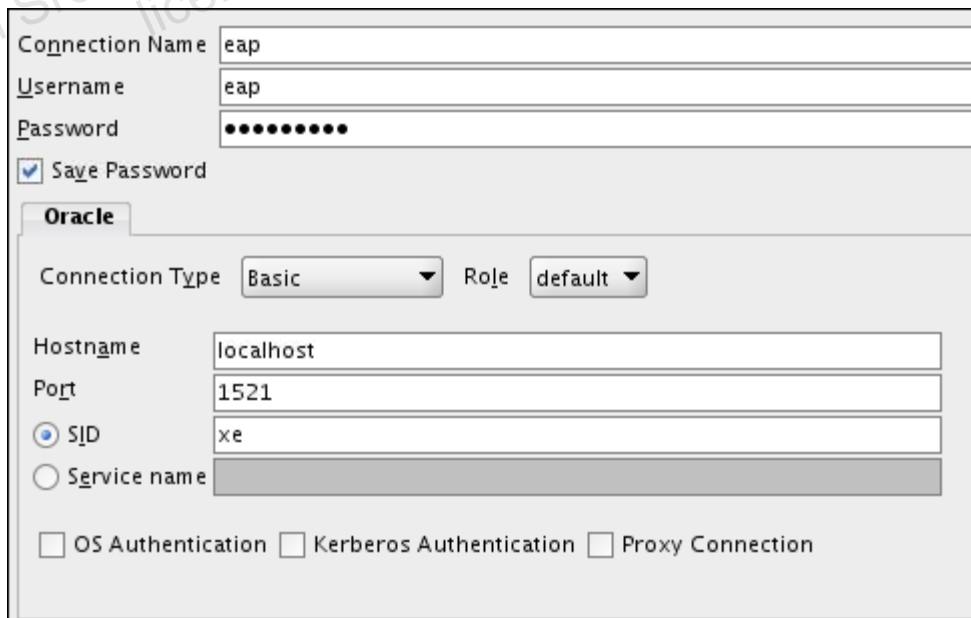
(Optional) View the Data

Your training environment comes with Oracle SQL Developer installed. This is a graphical user interface and is one way to view the Dynamusic data in the database. If you are not familiar with the tables or would like to view them in this UI, you may want to connect to SQL Developer and look at them.

20. Open a terminal window and enter `sqldeveloper` on the command line. **TO HERE**

21. To create a new connection, click the plus  icon. Enter the following values, select to save the password, and click **Connect**.

- Connection name: eap
- Username: eap
- Password: password1
- Hostname: localhost
- Port: 1521
- SID: xe



Connection Name: eap

Username: eap

Password:

☒ Save Password

Oracle

Connection Type: Basic Role: default

Hostname: localhost

Port: 1521

☒ SID: xe

☐ Service name

☐ OS Authentication ☐ Kerberos Authentication ☐ Proxy Connection

22. Under the connection for eap, 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 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.) You can view different information about a table, such as the columns, the data, and more through the various tabs available to you.

The screenshot shows the Oracle SQL Developer interface. On the left, the 'Connections' pane is open, showing a tree of database objects under the 'eap' connection. The 'DYNAMUSIC_ALBUM' table is selected. On the right, the 'Data' tab is active, displaying the table's contents. The table has three columns: 'ID', 'TITLE', and 'COVER'. The data is as follows:

ID	TITLE	COVER
1 500135	A Window to You: A Tribute to Buster Smith	(null)
2 300093	Bring On The Night	images/BringOnTheNig
3 300092	The Wolf	images/TheWolf.jpg
4 300091	Live In Montreal	images/LiveInMontrea
5 300132	The First Album	images/TheFirstAlbum
6 300131	No Regrets	images/NoRegrets.jpg
7 300133	Miami Nights	images/MiamiNights.j
8 300041	Eight Days	images/EightDays.jpg
9 300042	Red Flag	images/RedFlag.jpg

Practice 1-2: Creating a Custom Component

Overview

In this practice, you create a custom component class.

Over the course of the next few lessons, you write a custom process for deleting an artist from the repository. Deleting a repository item does not usually require custom processing, but you may recall that `artist` repository items do not include a list of albums. Instead, albums have a pointer to the artist. Deleting all albums associated with a particular artist therefore requires custom processing. The components and pages that you build over the next three tasks enable you to delete an artist and all of its associated albums in a single form entry. The first step in this process is creating the skeleton of a custom Songs (Repository) Manager component that is augmented and used in the practices of upcoming lessons.

Assumptions

Note that, in a real-world application, code would need to be written to remove all repository references to the deleted artist item. For Dynamusic, for example, that includes the `artists` property of the `concert` item type in the Events Repository. To simplify this task, you will be concerned only with cleaning up the `SongsRepository`.

Tasks

Write a Custom `SongsManager` Java Class

1. You should still be in the Java EE view in Eclipse. If not, choose **Window > Open Perspective > Other** and select Java EE. Create a new class in Eclipse by selecting **File > New > Other**. Under the Java menu, choose **Class** and click **Next**.
2. In the New Java Class wizard, put the class in the `dynamusic` package and give it the name `SongsManager`. Click the **Browse** button for the superclass and guide the wizard to find the appropriate class by entering “GenericServ” (not case-sensitive) in the box at the top. From the narrowed list, select `atg.nucleus.GenericService`. Back in the New Java Class wizard, select **Constructors from Superclass**, and then click **Finish**.
3. Eclipse creates a skeleton class file for you. In your class you should:
 - Import `atg.repository.*` for readability
 - Create a private member variable of type `atg.repository.Repository`. Add `getRepository` and `setRepository` methods to return and set the variable.

Hint: Using Eclipse, if you define a member variable `repository`, you may then select it and choose **Generate Getters and Setters** from the **Source** menu to automatically generate the code.

- Include a method with the following signature. For now, just create a stub for this method. You write meaningful code for it in a subsequent practice.

```
public void deleteAlbumsByArtist(String pArtistId)
    throws RepositoryException
```

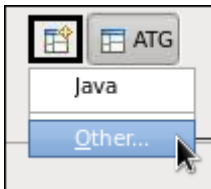
- Insert a `logDebug()` output line in each of your methods, which logs an indication that the method has been called. Make sure that your `logDebug()` lines are displayed only if `isLoggingDebug()` returns true.
- You do not need to provide any code for the constructor.

4. Save your file. Eclipse automatically compiles the class. Be sure to check for errors.
5. Build your project in Eclipse by selecting **Build All** from the **Project** menu. You need to build your project before you can use a new class in a component. You can skip this step in future exercises by selecting **Build Automatically** from the **Project** menu. Select **Build Automatically** now.

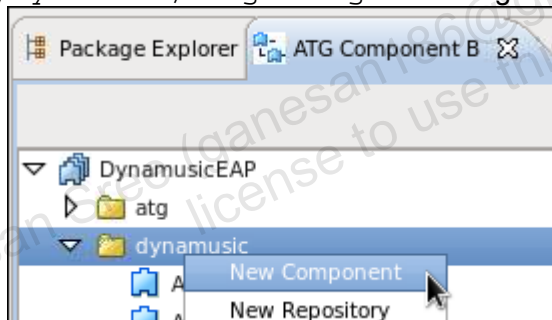
Note: You must build your project before you can use your class in an ATG component.

Create a SongsManager Component

6. Go to the Component Browser panel by clicking the Open Perspective button, selecting Other, and selecting ATG or by going to **Window > Show View > ATG Component Browser**.



Click the ATG Component Browser tab and create a new component called `/dynamusic/SongsManager` that is global in scope, based on your new class.



Select **Edit component after wizard completion** and click **Finish**.

New Component
Define the component.

Classpath Info
Module:

Component Info
Scope:

Class name: (e.g., atg.nucleus.InitialService)

Component name: (e.g., /atg/Initial)

☒ **Edit Component after wizard completion**

7. Set the following properties and save the component:
loggingDebug: **true**
repository: **/dynamusic/SongsRepository**.
8. You must restart ATG. On the Ant tab, double-click the **deploy** task. You can follow the process by looking at the messages on the Console tab. The deploy task automatically restarts ATG. You may proceed after you see the "Nucleus running" message.

Note: You can also restart your ATG server by pressing Ctrl + C in the server window and then double-clicking **startATGServer** in the EAP_Shortcuts folder. The practices use the deploy task.

Test Your Custom SongsManager Component

9. A test page called `songsManagerTest.jsp` has been provided for your use in testing the SongsManager component. `songsManagerTest.jsp` accepts an input parameter called `artistId` and attempts to delete the artist's albums via the `deleteAlbumsByArtist` method of SongsManager. The results of the attempt are rendered to the page.

Load `songsManagerTest.jsp`, passing in an artist ID (`artistId`) of your choosing. Remember, you can use the ACC to determine the ID of an artist. The URL should look like the following (with whatever artist ID you choose):

`http://localhost:7103/dynamusic/songsManagerTest.jsp?artistId=100004`

10. Verify that the rendered page indicates that your albums have been deleted. Remember, of course, that your albums have not really been deleted yet. So far, you have only provided a stub for the `deleteAlbumsByArtist` method. You write the actual delete album code in the practice for the next lesson.
11. Examine the ATG debug messages, either in the command window or in the log file (`<atgdir>/home/logs/debug.log`) to verify that the appropriate debug output you coded for the SongsManager component is displayed.

Practices for Lesson 2: The Repository API

Chapter 2

Practices for Lesson 2: Overview

Practices Overview

In these practices, you use the Repository API to access and modify repository data.

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

Practice 2-1: Catching Up

Overview

You must complete the practices from the previous lesson to perform the practices for the rest of the course. If you were unable to complete the previous practices, complete this practice. **If you finished the practices for the previous lesson, skip this practice and proceed to Practice 2-2.**

Tasks

1. If you have not done so yet, complete the steps in Practice 1-1 to set up your Java environment. To make things simpler, you can copy the following file from the Lesson 1 solutions folder (`<ourcedir>/solutions/lesson01`):
 - Copy the `MANIFEST.mf` file to `<atgdir>/DynamusicEAP/META-INF` (step 1).
2. Copy the solutions files from the Lesson 1 solutions folder (`<ourcedir>/solutions/lesson01`):
 - Copy the properties file to `<atgdir>/DynamusicEAP/config/dynamusic`.
 - Copy the Java file to `<atgdir>/DynamusicEAP/src/dynamusic`. This places the code in the correct directory for use in the following practices.
3. Build your project in Eclipse by selecting **Build All** from the **Project** menu. You can skip this step in future exercises by selecting **Build Automatically** from the **Project** menu.
Note: You must reassemble your EAR file and deploy it to WebLogic before the changes become active on your ATG server. You do this as part of Practice 2-2.

Practice 2-2: Using the Repository API

Overview

Implement the `SongsManager.deleteAlbumsByArtist` method: Delete all albums by the artist whose ID is passed into the method.

Tasks

Implement “Albums by Artist” Deletion

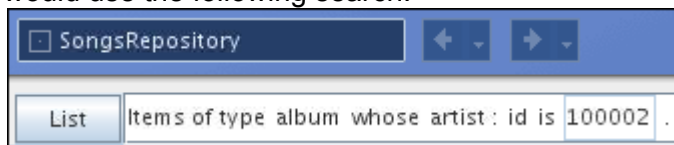
1. In `SongsManager.java`, import `atg.repository.rql.*`.
2. To the `deleteAlbumsByArtist` method, add code that uses the Repository API to find and delete all albums (album repository items) that are associated with an artist whose artist ID is specified by a given `String` input parameter. Your code should use an RQL query to retrieve all albums for an artist (that is, all albums whose artist's `id` property matches the ID passed in to the method), and then loop through and delete each album. After an album has been deleted, be sure not to use the object pointer referring to it.
Note: You can refer to the slides and the *ATG API Reference* for help. Include the logging debug code for each deleted album, printing out the album's ID.
3. Save `SongsManager.java`. After Eclipse auto-compiles, you should check for errors.
4. Deploy your changes by using the deploy task on the Ant tab.

Test Your Complete Songs Manager Component

5. Reload `songsManagerTest.jsp`, passing in an artist ID (`artistId`) of your choosing, such as:
`http://localhost:7103/dynamusic/songsManagerTest.jsp?artistId=100002`

Verify that the rendered page indicates that your artist's albums have been deleted. This time, if the `deleteAlbumsByArtist` method has been coded correctly, all the artist's albums should be deleted from the `SongsRepository`.

6. Start the ACC by visiting `http://localhost:7103/dyn/admin`. Log in as the user `admin` (password `password1`). Click the **ATG Control Center Administration** link, click the **Start ACC in Server VM** button, and log in as the user `admin` (password `password1`).
7. Use the ACC to verify that all the artist's albums have been deleted by choosing **Content > SongsRepository** and searching for albums whose artist ID matches the ID you entered in `songsManagerTest.jsp`. For example, using the artist ID from the step 5 example, you would use the following search:



The screenshot shows a web interface for the ATG Control Center. At the top, there is a search bar with the text "SongsRepository" and two navigation buttons (left and right arrows). Below the search bar, there is a "List" button and a text input field containing the search query: "Items of type album whose artist: id is 100002".

8. Examine the ATG debug log to verify that appropriate debug output messages coded for the `SongsManager` component (including the entry for each individual album deletion) are displayed.

Practices for Lesson 3: Custom Form Handlers

Chapter 3

Practices for Lesson 3: Overview

Practices Overview

In these practices, you create a custom form handler that extends `RepositoryFormHandler` to add additional application-specific functionality. You also create a custom form handler based on the `GenericFormHandler` base class.

Practice 3-1: Catching Up

Overview

You must complete the practices from the previous lessons to do the practices for the rest of the course. If you were unable to complete the previous practices, complete this practice. **If you finished the practices for the previous lessons, skip this practice and proceed to Practice 3-2.**

Tasks

1. Complete the “Catching Up” instructions from Practice 2-1 if you did not complete the practices for Lesson 1.
2. Copy the solutions files from the Lesson 2 solutions folder (`<ourcedir>/solutions/lesson02`):
 - Copy the properties file to `<atgdir>/DynamusicEAP/config/dynamusic`.
 - Copy the Java file to `<atgdir>/DynamusicEAP/src/dynamusic`. “Refresh” the Dynamusic project in Eclipse to recompile.

Note: You must reassemble your EAR file and deploy it to WebLogic to make the changes active on your ATG server. You do this as part of Practice 3-2.

Practice 3-2: Extending the Repository Form Handler

Overview

Create a class that extends `RepositoryFormHandler` to handle the deletion of artists. Make sure that all albums associated with an artist are deleted when that artist is deleted.

Tasks

Create a Custom Artist Form Handler Class

1. Create an `ArtistFormHandler` Java class (`ArtistFormHandler.java`) in the `src/dynamusic` folder of the `Dynamusic` module. Use the following guidelines when writing your class:
 - Your class should reside in a package called `dynamusic`.
 - Your class should extend the `atg.repository.servlet.RepositoryFormHandler` class.
 - For readability, import the following packages:
`atg.droplet.*`
`atg.repository.*`
`atg.repository.servlet.*`
`atg.servlet.*`
 - Include `get` and `set` methods for a property called `songsManager`. The `songsManager` property is of type `dynamusic.SongsManager`. (**Hint:** Define a member variable and use the “Eclipse Generate Getters and Settings” facility.)
 - Override the `preDeleteItem()` method of `RepositoryFormHandler`. Add code to call the `deleteAlbumsByArtist()` method of the component pointed to by the `songsManager` property. Also, be sure to call the `preDeleteItem()` method of the super class. (**Hint:** use the **Source > Override Implement Methods** Eclipse menu option.)
 - The signature for `preDeleteItem()` is:

```
protected void preDeleteItem(DynamoHttpServletRequest  
pRequest,DynamoHttpServletResponse pResponse)  
throws javax.servlet.ServletException,java.io.IOException
```

When writing the code for the method, remember that the `RepositoryFormHandler` class (which you are extending) provides you access to the repository item and the item’s ID. Review the *ATG API Platform Reference* for the methods to retrieve those values. (The values are set by the form JSP.)
 - Include `logDebug()` output lines in your method to assist in debugging your form handler.

Modify the Artist Form Handler Component

2. Modify the `ArtistFormHandler` component to use your custom `ArtistFormHandler` class by editing the `.properties` file and restarting ATG. (You cannot change the class of an existing component by using the Component Editor in Eclipse.) The `ArtistFormHandler` component was previously of type `atg.repository.servlet.RepositoryFormHandler`, which is the base class of the

`dynamusic.ArtistFormHandler` class. So you are simply extending your component to include the new “delete albums by artist” functionality while preserving all other methods.

3. Set the new `songsManager` property of the `ArtistFormHandler` component to `/dynamusic/SongsManager`.
4. Deploy your changes by using the deploy task on the Ant tab.

Test the Artist Form Handler

5. In the *Foundations of ATG Application Development* course, you created a page to edit artist information. This page contained a “Delete this Artist” button. This button now activates your custom `preDeleteItem()` method and initiates the deletion of all albums by the artist.
6. Verify this by selecting an artist, opening the artist details page, clicking the Edit Artist Info link, and clicking the **Delete this Artist** button.
7. Load the Artists page and verify that your artist is no longer listed. Use the ACC to verify that all of that artist’s albums have also been deleted.
8. Examine the ATG server window. Verify that this window displays appropriate debug messages that are coded for the `ArtistFormHandler` and `SongsManager` components.

Practice 3-3: Creating a Custom Form Handler

Overview

Create a custom form handler that implements a Dynamusic trivia question. This form handler enables an administrator to configure the component with a question and an answer. The user can enter an answer, and the form determines if the user's answer matches the configured answer. The form handler should do error handling and redirection as configured from the form.

Tasks

Create a Custom `QuizFormHandler` Class

1. Create a `QuizFormHandler` Java class. Use the following guidelines when writing your class:
 - Your class should reside in a package called `dynamusic`.
 - Your class must extend the `atg.droplet.GenericFormHandler` class.
 - Include `get` and `set` methods for properties called `question`, `answer`, and `userAnswer`. These properties should be of type `String`. (Remember to save the property values as private member variables.)
 - Include methods for a `correctAnswer` property of type `boolean`. Implement an `isCorrectAnswer` method instead of `getCorrectAnswer`.
 - Include `get` and `set` methods for properties of type `String` called `validateSuccessURL` and `validateErrorURL`.
 - Implement a `handleValidate` method that compares the `userAnswer` property to the `answer` property (ignoring case!) and sets the `correctAnswer` boolean property appropriately. The method should then redirect (`response.sendRedirect()`) to the `validateSuccessURL` (if it is not null). The validation code should be preceded by a check to see if any form errors have occurred and, if so, redirect to the `validateErrorURL` (if it is not null).

Note that an incorrect answer by the user is *not* considered to be an error. As long as no exceptions or errors occur, the form should redirect to the success URL. (The success JSP page can display the results of the quiz—whether the answer was correct or not. The error JSP page displays an error message, such as *no answer was entered*.) Remember that the `handle` method must return `false` if you redirect within the method, and `true` otherwise.

The signature for `handleValidate()` is as follows:

```
public boolean handleValidate(DynamoHttpServletRequest
    request, DynamoHttpServletResponse response)
    throws java.io.IOException
```

- Implement a `handleCancel` method that sets `userAnswer` to null.
- Include `logDebug` output in the `handle` method and property set methods.

Create a Quiz Form Handler Component

2. Create a global scoped `/dynamusic/QuizFormHandler` component based on `dynamusic.QuizFormHandler`. Set the `question` and `answer` properties to a music trivia question and answer of your choosing. Turn on debugging for the component for testing.

Create Quiz and Quiz Answer Pages

3. Create a page called `quiz.jsp` that implements a form to ask the user the predefined trivia question and accept a text answer. Your form should be processed by your custom `QuizFormHandler` component with the text input field attached to the `userAnswer` property (which should be marked `required`). Set the success URL to a page called `quizAnswer.jsp`, which you create in step 5.
4. Include error handling by using `ErrorMessageForEach` on `quiz.jsp`.
5. Create a page called `quizAnswer.jsp` that checks the results (using a `Switch` servlet bean switching on the `correctAnswer` Boolean property) of the quiz answer validation and renders an appropriate message.
6. Deploy your changes by using the deploy task on the Ant tab.

Test the Quiz Form Handler

7. Temporarily turn on debugging for the new `QuizFormHandler` component in the Dynamo Administration. To do this, go to `http://localhost:7103/dyn/admin`, log in as the user `admin` with the password `password1`, click the **Component Browser** link, and navigate to your component. Click the **loggingDebug** property, select **true**, and click the **Change Value** button.
8. Load `dynamusic/quiz.jsp`, answer your question correctly, and verify your quiz form handler class, component, and pages.
9. Reload `dynamusic/quiz.jsp`, test with a wrong answer, and verify your processing of an incorrect answer.
10. Reload `dynamusic/quiz.jsp` and submit the form with no data for the answer.
11. Review the debug log file for debug output.

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

Practices for Lesson 4: Managing Transactions

Chapter 4

Practices for Lesson 4: Overview

Practices Overview

In these practices, you use the Transaction API to control transactions.

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

Practice 4-1: Catching Up

Overview

You must complete the practices from the previous lessons to do the practices for the rest of the course. If you were unable to complete the previous practices, complete this practice. **If you finished the practices for the previous lessons, skip this practice and proceed to Practice 4-2.**

Tasks

1. Complete the “Catching Up” instructions from Practice 3-1 if you did not complete the practices for Lesson 2.
2. Copy the solutions files from the Lesson 3 solutions folder (`<ourcedir>/solutions/lesson03`):
 - Copy the properties files to `<atgdir>/DynamusicEAP/config/dynamusic`.
 - Copy the Java files to `<atgdir>/DynamusicEAP/src/dynamusic`. “Refresh” the Dynamusic project in Eclipse to recompile.
 - Copy the JSP files to `<atgdir>/DynamusicEAP/j2ee-apps/dynamusic-j2ee/dynamusic-web.war`.

Note: You must reassemble your EAR file and deploy it to WebLogic to make the changes active on your ATG server. You do this as part of Practice 4-2.

Practice 4-2: Controlling Transactions

Overview

Recall that the `ArtistFormHandler` component first deletes all of an artist's albums and then deletes the artist itself. What if one or all of the album deletion attempts fail for some reason? In such a case, you might want to guarantee that the artist is not deleted with one or more "orphaned" albums remaining in the repository. Or, conversely, if the deletion of the artist fails, you might want to roll back the deletion of the albums.

Fortunately, `ArtistFormHandler` is based on `RepositoryFormHandler`, which already handles transactions. If any portion of the transaction fails (including the `preDelete` and `postDelete` methods), the entire transaction rolls back. Unfortunately, however, the `SongsManager` provides the `deleteAlbumsByArtists` method without knowing which class will be calling it or whether that class already does intelligent transaction management. So you must wrap the album deletion in a transaction to guarantee that if any of the album deletion attempts fail, all will fail, and the entire transaction will roll back.

Tasks

1. Modify the `SongsManager.java` file. Add a `get` and `set` method for a property called `transactionManager`, of type `javax.transaction.TransactionManager`. (You may want to import `javax.transaction.*` and `atg.dtm.*` for readability.)
2. In the `deleteAlbumsByArtist` method, put transaction code around the deletion of albums. Use a transaction demarcation of type `REQUIRED` so that, when the code is called when an existing transaction is in place (such as from a `RepositoryFormHandler` component), the album deletion simply joins the existing transaction. Otherwise, a new transaction is created.
3. If any exceptions occur during the deletions, catch them and call `setRollbackOnly()` on the transaction manager.
4. When you are finished, save your code and make sure that it compiles cleanly.
5. Modify the `/dynamusic/SongsManager` component, linking the new `transactionManager` property to the `/atg/dynamo/transaction/TransactionManager` component.
6. Deploy your changes by using the `deploy` task on the Ant tab.
7. Make sure that `SongsManager` still works as expected, using either the `songsManagerTest.jsp` page or the `editArtist.jsp` page.
8. Transactions are relevant only if a problem occurs during repository access, which is unlikely in code that you have written in such a straightforward way. To test your code, you may want to create an artificial "exception" to see what happens. Try adding code *after the albums have been deleted* within your transaction demarcation to throw an exception such as:

```
throw new RepositoryException("this is just a test");
```


Reassemble your EAR and deploy it to WebLogic. Test deleting an artist, and make sure that the transaction rolled back and that neither artist nor albums were deleted.

9. If your transactions are not working as expected, or if you want to confirm exactly how they are working, you can modify the *live* property values of the `/atg/dynamo/transaction/TransactionManager` component, setting `loggingDebug` and `loggingTransactions` to `true`.

Note: Recall that, to set live component values, you can use either the ACC or the Admin UI Component Browser (<http://localhost:7103/dyn/admin>). Because this generates a large amount of output, be sure to set these values back to `false` before continuing with the practices. Also be sure to remove your exception simulation code.

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: Complex Repository Configuration

Chapter 5

Practices for Lesson 5: Overview

Practices Overview

In these practices, you configure a many-to-many relationship between repository items.

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

Practice 5-1: Catching Up

Overview

You do not need to complete the practices from the previous lessons to do this practice. However, if you were unable to complete the previous practice and want to catch up, complete this practice. **If you finished the practices for Lesson 4, skip this practice and proceed to Practice 5-2.**

Tasks

1. Copy the solutions files from the Lesson 4 solutions folder (`<course_dir>/solutions/lesson04`):
 - Copy the properties file to `<atg_dir>/DynamusicEAP/config/dynamusic`.
 - Copy the Java file to `<atg_dir>/DynamusicEAP/src/dynamusic`. “Refresh” the Dynamusic project in Eclipse to recompile.

Note: You must reassemble your EAR file and deploy it to WebLogic to make the changes active on your ATG server. You do this as part of Practice 5-2.

Practice 5-2: Configuring a Many-to-Many Relationship

Overview

In the current `SongsRepository`, there is a one-to-many relationship defined between artists and albums. That is, each album can have one artist, and artists can have any number of albums. This relationship is expressed through a single property, `album.artist`.

Dynamusic needs to define a more complex relationship between albums and artists. An album may have more than one artist (for example, a compilation album or anthology album), *and* an artist can have more than one album. Implement this many-to-many relationship by adding two new properties:

- **album.artists:** A property on the `album` item type containing a set of all the artists for the album
- **artist.albums:** A property on the `artist` item type containing a set of albums released by the artist.

Note: The legacy `album.artist` property is left in the application for backward compatibility.

Assumptions

The SQL table required to implement this relationship has already been created in the database, as described by the following SQL command:

```
CREATE TABLE dynamusic_artist_album (  
  album_id VARCHAR(32) not null references dynamusic_album(id),  
  artist_id VARCHAR(32) not null references dynamusic_artist(id),  
  primary key(album_id, artist_id)  
);
```

You populate this table with data from the `artist` column of the `dynamusic_album` table as part of this practice.

You edit the XML definition file directly

(`<atgdir>/DynamusicEAP/config/dynamusic/songs.xml`) using the Eclipse XML editor. Refer back to the slides for guidance on which tags and attributes to add.

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 about using the Eclipse Web Tools XML editor, see **Help > Help Contents > Web Tools Platform User Guidet > Editing Markup Language Files**.

Tasks

Map the Artists and Albums Properties

1. Go to the Package Explorer tab and navigate under the `DynamusicEAP` project to `config/Dynamusic`. Double-click `songs.xml` to open the file. You will be viewing the repository in the Design view. Click the Source tab to see the XML source code in the file. Add a new table definition to the `artist` item descriptor with the name to `dynamusic_artist_album`. Add attributes to the table tag as follows: Set the `id-column-name` attribute to `artist_id` and the type to `multi`.
2. Create a property called `albums` with a data-type of `set` that maps to the `album_id` column. Add two more attributes to `set` component - `item-type` to `album` and `column-name` to `album_id`.

3. Repeat the previous two steps, substituting the appropriate property name and attribute values, to create an `artists` property on the `album` item descriptor.
4. Save your changes.
5. Test your work by using `startSQLRepository`. In a terminal, change directory to `<atgdir>/home/bin`, and enter the following two commands:

```
startSQLRepository -m DynamusicEAPTest -repository
/dynamusic/SongsRepository
```

 Resolve any errors or warnings before proceeding. Leave this window open so that you can use it for testing in the practices for the lesson titled “Advanced SQL Repository Configuration.”

Load Data and Test the Repository Extensions

6. Deploy your changes by using the deploy task on the Ant tab.
7. You now need to populate your `album.artists` and `artist.albums` properties so that you have some data to easily look at. You do this through the Dynamo Admin JDBC Browser. In your browser, go to Dynamo Administration (<http://localhost:7103/dyn/admin/>) and log in (`admin/password1`) if necessary. Click the **JDBC Browser** link, and then click the **Execute query** link. Enter the following SQL in the query field:

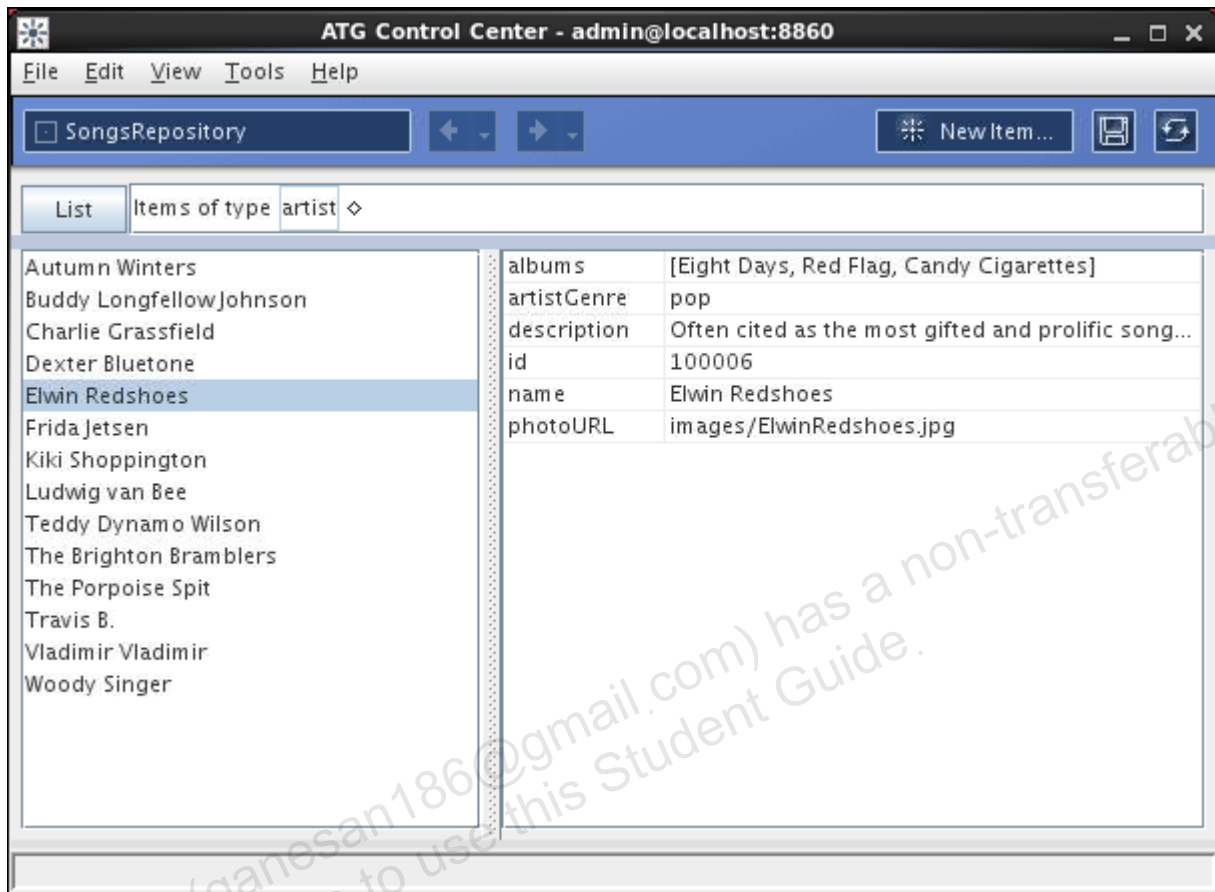
```
insert into dynamusic_artist_album (album_id, artist_id) select id,
artist FROM dynamusic_album
```

Click the **execute and commit** button.

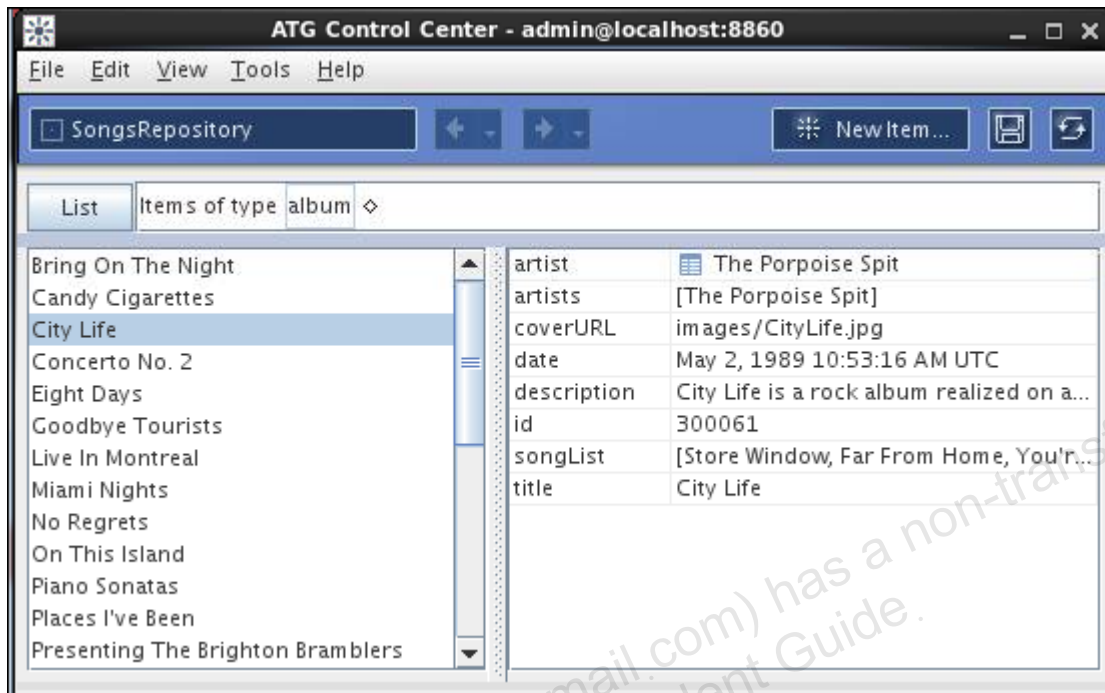
Note: If the server had cached any of your `album` or `artist` items prior to this step, those items would not reflect the above change until the cache was invalidated. Repository caching is the topic of a later lesson.

8. Start the ACC. In the ACC, select **Content > SongsRepository**.

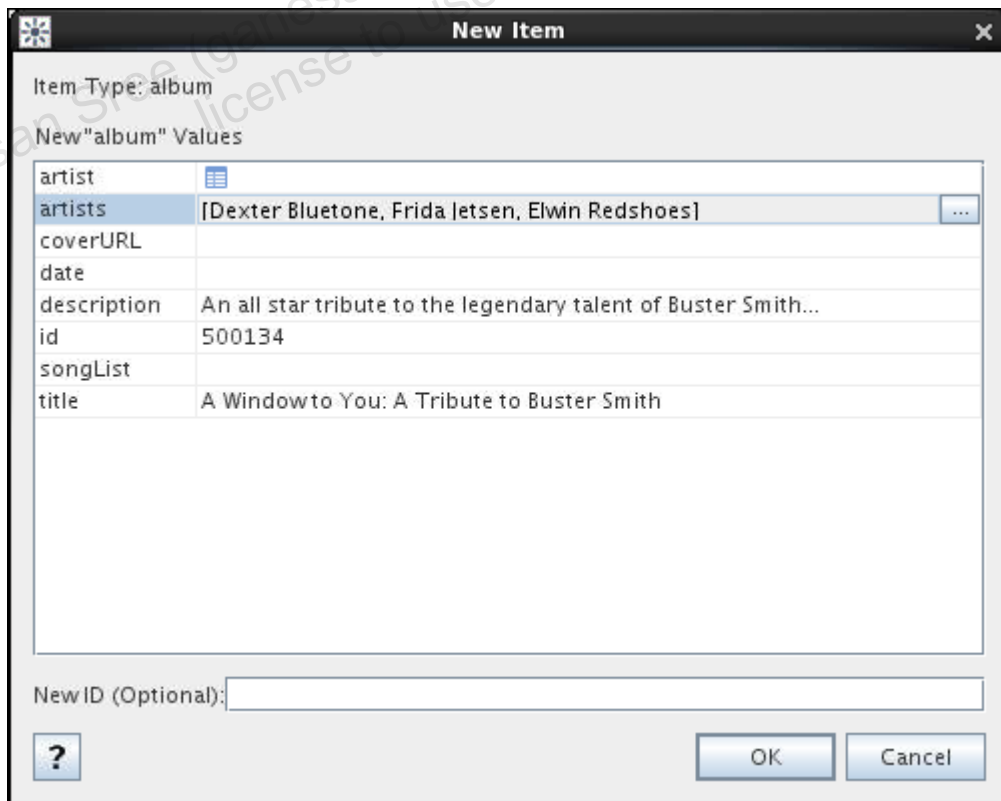
9. List items of type `artist`. Select an artist and verify that the `albums` property is listed and that it contains a list of that artist's albums.



- List items of type album. Select an album and verify that the `artists` property is listed and that it contains a list of that album's artists. At this time, that list should contain a single artist per album.



- Add a new album item, specifying multiple artists for the `artists` property. Verify that the `artists` property contains the list of artists.



Tip: The album does not need to have songs for this test. If you choose to add songs, be aware that Dynamusic has chosen to model the data in such a way that each song can be on only one album. If you try to add an existing song to a second album, you receive a message about existing references, and a new copy of that song is created.

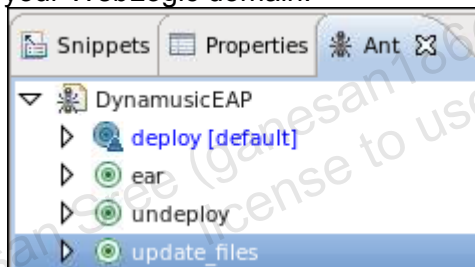
Use a Many-to-Many Relationship in a JavaServer Page

12. When you are planning your application, you typically create the correct one-to-many and many-to-many relationships upfront. In this course, you are in the unusual position of modifying code previously created for a different data model.

To support the change from the single-value `artist` property to the multi-value `artists` property on the `album` type, some changes to the existing JSPs need to be made. To save you some time, those modifications have been provided for you. You use these modified files to test your item type extensions.

Copy the JSP files in `<ourcedir>\src\lesson05\` to the `<atgdir>\DynamusicEAP\j2ee-apps\dynamusic-j2ee\dynamusic-web.war\` folder, replacing the existing JSPs. Examine the files listed above and make sure that you understand the changes that were made to utilize your new properties. To trigger a recompile of these pages, make a small change in a text editor (such as adding a space) and save each file. This updates the last modified date of the file.

13. On the Ant tab, double-click **update_files**. This copies your files to the document root for your WebLogic domain.



14. In your browser, open the Dynamusic artists page (<http://localhost:7103/dynamusic/artists.jsp>) and select one of the artists that you assigned to the multi-artist album you created earlier. Verify that your multi-artist album is listed.
15. Select your multi-artist album. Verify that all the artists that you assigned to the album are listed.
16. Select each of the other artists listed for your album. Verify that your multi-artist album is listed for each artist.

(Optional) Use a Many-to-Many Property in Place of RQL Query

17. Edit `artistDetails.jsp` and replace the `RQLQueryForEach` servlet bean that uses the artist ID to search for related albums in the `SongsRepository` with a `ForEach` servlet bean that simply uses the multi-value `albums` property of `artist` to list the associated albums.
18. Load an artist page and verify your work.

Practices for Lesson 6: Advanced SQL Repository Configuration

Chapter 6

Practices for Lesson 6: Overview

Practices Overview

In these practices, you will create custom property types, then add attributes for controlling the display of the property value.

Practice 6-1: Catching Up

Overview

You do not need to complete the practices from the previous lessons to do this practice. However, if you were unable to complete the previous practice and want to catch up, complete this practice. **If you finished the practices for Lesson 5, skip this practice and go on to Practice 6-2.**

Tasks

1. Copy the solutions files from the Lesson 5 solutions folder (`<ourcedir>/solutions/lesson05`):
 - Copy the JSP files to the `DynamusicEAP\j2ee-apps\dynamusic-j2ee\dynamusic2.war` folder.
 - Copy `songs.xml` to `DynamusicEAP\config\dynamusic`.

Note: You must reassemble your EAR file and deploy it to WebLogic to make the changes active on your ATG server. You do this as part of Practice 6-2.

Practice 6-2: Creating a Custom Property

Overview

In this practice, you create a custom property that calculates the user's age from the user's birthdate. You also create attributes that show the user's age in years or in days.

Tasks

Write a Java Class That Implements a New Transient Custom Property That Calculates the User's Age

1. Write a `FindAge` Java class (`FindAge.java`) and save it in the `src/dynamusic` folder of the `DynamusicEAP` module. Use the following guidelines when writing your class (for guidance, refer to the example code covered in the lesson):
 - Your class should reside in a package called `dynamusic`.
 - For readability, import `atg.repository.*`. Import `java.util.Date`.
 - Your class must extend the `atg.repository.RepositoryPropertyDescriptor` class.
 - Declare the `String` member variable `mAgeFormat`, which has two possible values: `years` and `days`.
 - Declare a constant called `AGE_FORMAT_ATTR_NAME` that is set to `ageFormat`. This is the name of the attribute that is set in the XML repository definition file to control the age format. Remember that a constant is defined as a `public static final` member variable.
 - Override `setValue()` to allow for the reading of the `ageFormat` attribute value from the repository definition file. (This method is called automatically by the repository definition code when an attribute is set in the definition file. By overriding it, you can check whether your custom attribute has been set. Remember that the name of your custom attribute is stored in the `AGE_FORMAT_ATTR_NAME` constant.) Set the `mAgeFormat` member variable to either `years` or `days` based on the value supplied for the `AGE_FORMAT_ATTR_NAME` attribute.
 - Override `getPropertyValue()` to calculate the user's age from the `dateOfBirth` property. It should return age in `years` or `days`, depending on the value of `mAgeFormat`. Make `years` the default. When you get the `dateOfBirth` property, it will be of type `Object`. You should cast it to `java.util.Date`.

To assist you in performing the calculations, you have been provided with a utility class called `dynamusic.AgeCalc` with the following public static methods:

`public static int ageInDays(Date pDate):` Returns the number of days since the date specified

`public static int ageInYears(Date pDate):` Returns the number of years since the date specified

Note that `getPropertyValue()` returns an `Object` type, whereas the utility class returns a primitive (`int`). Therefore, your code should convert the `int` primitive value to an `Integer` object by using the `Integer` object's constructor. An example is `new Integer(int)`.

- The property that you are creating will be transient. You should therefore make the property nonqueryable and nonwritable by overriding the `public boolean isQueryable()` and `public boolean isWritable()` methods to both return `false`.
- 2. Save your new class. When you save, Eclipse attempts to compile the class for you. Fix any compilation errors. Make sure that you get a clean compile before proceeding.

Add Your Custom Property to the User Profile

3. In Eclipse, switch to the **Package Explorer** tab in the upper-left corner. Navigate to the `atg/userprofiling/userProfile.xml` under `DynamusicEAP's config` folder. Double-click the file to edit it. It opens in the right pane. Switch from the **Design** tab to the **Source** tab.
4. Add a new transient property called `ageInYears` to the `user` item descriptor. The new property should be defined as follows:

- **data-type:** `int`
- **property-type:** `dynamusic.FindAge` (the class you just wrote)
- **default:** `0`
- **category:** `dynamusic`
- **writable:** `false`

Add an attribute tag object to the property to set the `ageFormat` attribute to `years`, with a `data-type` of `string`.

5. Create a second similar transient property called `ageInDays`, setting the `ageFormat` attribute to `days` instead.
6. In the test window you opened in the practices for the lesson titled “Complex Repository Configuration,” test your work by using `startSQLRepository`:

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

7. Deploy your changes by using the deploy task on the Ant tab. When the server is running again, open a browser to the Admin UI (<http://localhost:7103/dyn/admin>), click the ATG Control Center Administration link, and click the **Start ACC in Server VM** button (which sets the ACC's CLASSPATH appropriately). Test your changes by viewing a user profile in the ACC (select **People and Organizations > Users** to access the profiles) and then confirming that the new `ageInDays` and `ageInYears` properties are calculated correctly.

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

Practices for Lesson 7: Derived Properties

Chapter 7

Practices for Lesson 7: Overview

Practices Overview

In these practices, you configure a derived property and make it overrideable.

Practice 7-1: Catching Up

Overview

You do not need to complete the practices from the previous lessons to do this practice. However, if you were unable to complete the previous practice and want to catch up, complete this practice. **If you finished the practices for Lesson 6, skip this practice and proceed to Practice 7-2.**

Tasks

1. Copy the solutions files from the Lesson 6 solutions folder (`<ourcedir>/solutions/lesson06`):
 - Copy the `userProfile.xml` file to `<atgdir>/DynamusicEAP/config/atg/userprofiling`.
 - Copy the Java file to `<atgdir>/DynamusicEAP/src/dynamusic`. “Refresh” the Dynamusic project in Eclipse to recompile.

Note: You must reassemble your EAR file and deploy it to WebLogic to make the changes be active on your ATG server. You do this as part of Practice 7-2.

Practice 7-2: Creating a Derived Property

Overview

The artist item has an `artistGenre` property that reflects the primary genre of the music that an artist records. Usually, the genre of an album is the same as the primary genre of the artist. So you want the album to have a `genre` property whose default value is derived from the artist.

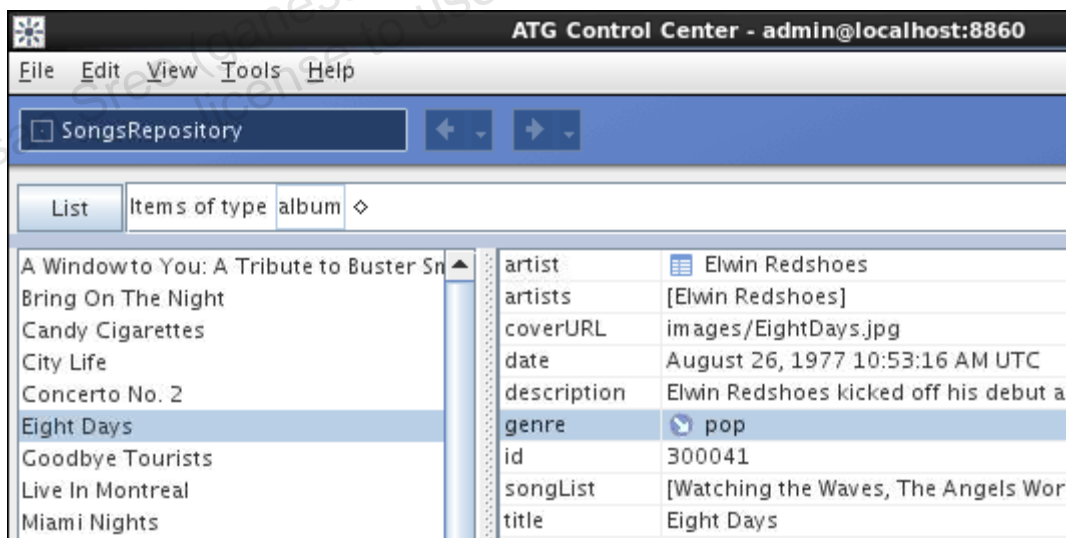
Assumptions

The rest of the practices for the course do not require Java coding. You still need to assemble your EAR and deploy it to WebLogic Server to make your XML changes appear on your ATG server.

Tasks

Add an Album Genre Property

1. Edit the `songs.xml` file in Eclipse.
2. Add a new transient property to the album item descriptor called `genre`. Set the property's data type to `string` and the `writable` attribute to `false`.
3. Configure the `genre` property as a property that derives from `artist.artistGenre`.
4. Test your work by using `startSQLRepository`.
5. Deploy your changes by using the deploy task on the Ant tab. Start the ACC in the server VM (as you have done in previous practices).
6. Use the ACC to examine an album and verify that the `genre` derived property is listed and set to the corresponding artist's genre.



Practice 7-3: Making a Derived Property Overrideable

Overview

Sometimes artists release albums outside their primary genre, so you should define the genre property to be overrideable. That is, you should create a second *writable* property (called `albumGenre`) and have the genre property use that property as its override property.

Assumptions

To implement the album genre, a column has been added to the primary table (`dynamusic_album`) for the album item type as follows:

```
CREATE TABLE dynamusic_album (  
    ...other columns...  
    genre NUMBER(10,0) null,  
    primary key(id)  
);
```

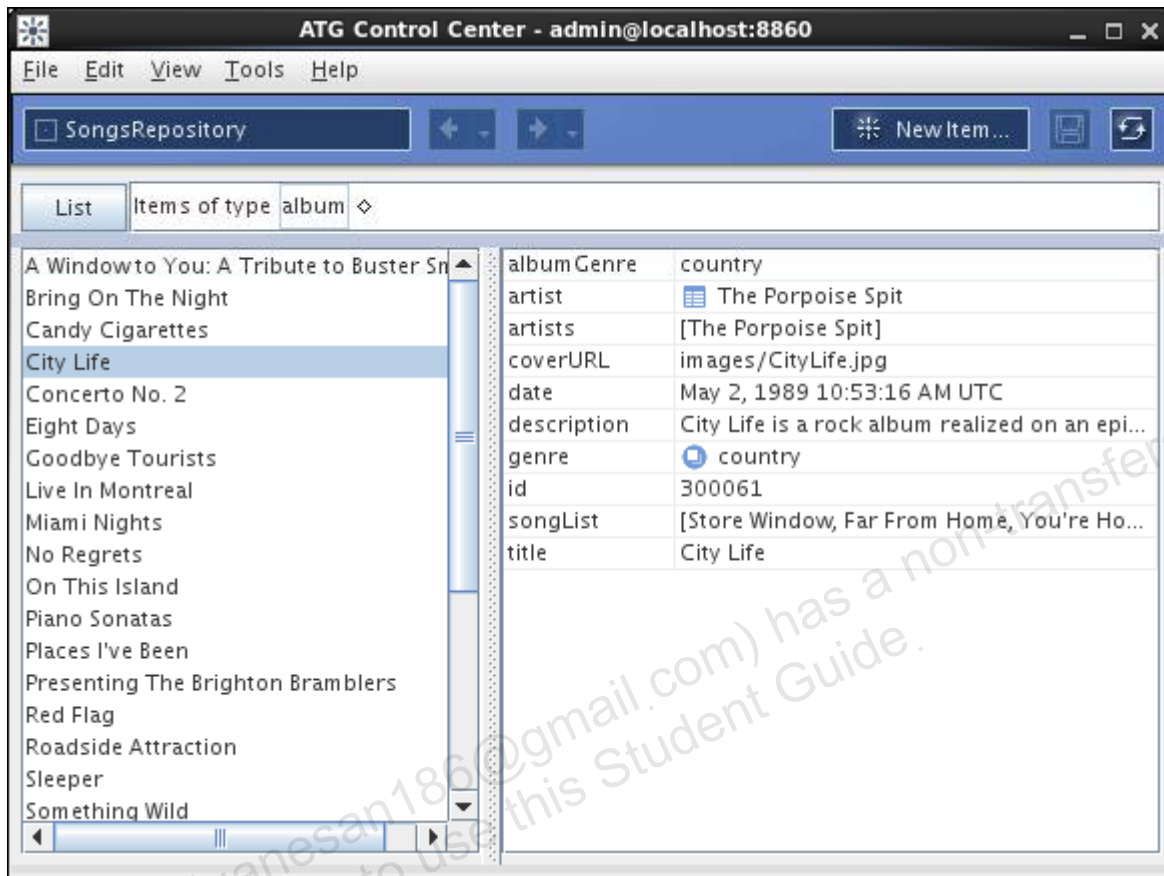
You usually hide an override property such as this so that it does not appear as a separate property in the ACC. In this case, you leave it visible to help with testing and debugging.

Tasks

Make the Genre Property Overrideable

1. Go to the Package Explorer tab and navigate under the DynamusicEAP project to `config/Dynamusic`. Double-click `songs.xml` to open the file. Under the album item descriptor, add a new property within the `dynamusic_album` table called `albumGenre`, which is mapped to the `genre` column. This property's data type should be `enumerated`, with the same options as the `artist.artistGenre` property. Create a `useCodeForValue` attribute set to `false`.
Hint: You can actually copy the `artistGenre` property definition and rename it to `albumGenre`.
2. Change the definition of the `album.genre` property that you created in the previous exercise so that the property is `writable` and its override property is `albumGenre`.
3. Test your work by using `startSQLRepository`.
4. Deploy your changes by using the deploy task on the Ant tab.

- Use the ACC to examine an album and verify that you can override the `genre` value and that the override value `country` is correctly stored in the `albumGenre` property.



Note: You may notice that you are able to enter values in `genre` that are not among the allowable options for the enumerated `albumGenre` property. Unfortunately, you cannot make derived properties enumerated, so the ACC cannot restrict users to only entering certain values. However, when you try to save the change, the ACC generates an error.

- Modify the `albumDetails.jsp` page to display the album genre property. Alternatively, if you are running short on time, an updated version of the page with the required modifications can be copied from `<coursedir>\src\lesson07\` to the Dynamusic web application directory.
- On the Ant tab, double-click the `update_files` task to run it. (If you closed your Ant tab, go to **Window > Show View > Other**, select **Ant** from below the Ant folder, and then click **OK**.)
- Load Dynamusic and browse to the details page of an album for which you have not overridden the genre, and then browse to another album for which you have overridden the genre. Verify that the correct genre is displayed.

Practices for Lesson 8: Repository Caching

Chapter 8

Practices for Lesson 8: Overview

Practices Overview

In these practices, you configure SQL repository caching and you review caching statistics.

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

Practice 8-1: Catching Up

Overview

You do not need to complete the practices from the previous lessons to do this practice. However, if you were unable to complete the previous practice and want to catch up, complete this practice. **If you finished the practices for Lesson 7, skip this practice and proceed to Practice 8-2.**

Tasks

1. Copy the solutions files from the Lesson 7 solutions folder (*coursedir>/solutions/lesson07*):
 - Copy the `songs.xml` file to `<atgdir>/DynamusicEAP/config/dynamusic.`
 - Copy the JSP file to `<atgdir>/DynamusicEAP/j2ee-apps/dynamusic-j2ee\dynamusic.war.`

Note: You must reassemble your EAR file and deploy it to WebLogic Server to make the changes active on your ATG server. You do not need the changes to do the practices in this lesson.

Practice 8-2: Preloading the SQL Repository Cache

Overview

In this practice, you do the following:

- Preload all the items in the `SongsRepository` when the repository starts up.
- Enable query caching on items in the `SongsRepository`.
- Review caching statistics.

Tasks

Preload the `SongsRepository` Cache

Preloading the cache involves querying the repository defined in a repository definition file by using XML repository operation tags.

1. Before setting up cache preloading, view the cache statistics. Open the **Dynamo Administration** interface at <http://localhost:7103/dyn/admin> and select the **Component Browser**. Browse to the `/dynamusic/SongsRepository` component and note the **Cache usage statistics** section of the page.
2. Click **Reset Cache Statistics** so that you see “fresh” statistics the next time you restart ATG.

Cache usage statistics							
Reset Cache Statistics							
	localEntries	externEntries	weakEntries	localCacheSize	usedRatio	totalHits	totalM
item-descriptor=song cache-mode=simple cache-locality=local							
Items 11	0	0	1000	1.1%	0	0	
item-descriptor=artist cache-mode=simple cache-locality=local							
Items 12	0	0	1000	1.2%	0	0	
item-descriptor=album cache-mode=simple cache-locality=local							
Items 29	0	0	1000	2.9%	0	0	

3. Edit the `songs.xml` file directly in Eclipse (or another text editor). At the end of the file, after all the `item-descriptor` tags but before the closing `gsa-template` tag, add a `query-items` tag for each of the three item types in the `SongsRepository`: `song`, `album`, and `artist`. The `query-items` tags generate a potentially huge amount of output on startup. To avoid this, you can add the `quiet="true"` attribute to the tags. For this practice, try adding `quiet="true"` to the `song` and `album` queries but not to the `artist` query so that you can become familiar with the debug output.
4. Test your repository modifications by using `startSQLRepository`. Note the query output. Resolve any errors before continuing.
5. Deploy your changes by using the deploy task on the Ant tab.

6. Return to the `SongsRepository` page in the **Component Browser**, and examine the **Cache usage statistics**.
7. Close out any open browser windows. In a new browser window, clear the cached pages in your browser. (For Firefox, you can do this by going to **Tools > Clear Recent History** (or pressing Ctrl + Shift + Delete) and then clicking the Clear Now button.) Browse through several artists, albums, and songs on the Dynamusic site. Reload the cache statistics page for `SongsRepository` in the Component Browser and note the cache hits.

Configure Query Caching

By default, query caching is disabled, as indicated on the **Cache usage statistics** page. You can enable it by setting the query cache size to a number greater than 0, which is the default value.

8. In each `item-descriptor` tag in `songs.xml`, add `query-cache-size="100"`.
9. Test your repository modifications by using `startSQLRepository`.
10. Deploy your changes by using the deploy task on the Ant tab.
11. Browse through several albums, artists, and songs on the site, and then reload the **Cache usage statistics** page. Note that query caches are no longer disabled. Instead, they appear and show several reused queries.

Note: Queries occur on several pages on the site. Look at `song.jsp` for an example.

Manually Invalidate the Cache

12. Go to the `SongsRepository` page in the **Component Browser**. Find the **Methods** section and locate the `invalidateCaches` method. Click the method name and then click the **Invoke Method** button to invoke it.

generateTableId	int	Class atg.adapter.gsa.GSARespository
hashCode	int	Class java.lang.Object
invalidateCaches	void	Class atg.adapter.gsa.GSARespository
invalidateExternalCaches	void	Class atg.adapter.gsa.GSARespository
notify	void	Class java.lang.Object

13. Reload the `SongsRepository` page and note that the caches are now empty.

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

Practices for Lesson 9: Custom Droplets

Chapter 9

Practices for Lesson 9: Overview

Practices Overview

In these practices, you create custom servlet beans.

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

Practice 9-1: Creating a Custom Servlet Bean

Overview

In this practice, you create a custom servlet bean that iterates through the possible values of an enumerated property and, for each value, returns the formatted HTML as defined by an `oparam`. You can use this servlet bean to create a form check box or select box for an enumerated property without having to explicitly list the enumerated options in the JSP that implements the form.

The `EnumForEach` servlet bean has the following parameters:

Input Parameters		
<code>repositoryName</code>	String	Full path name of the repository (such as <code>/dynamusic/SongsRepository</code>)
<code>itemDescriptorName</code>	String	Name of the item type (such as <code>album</code>)
<code>propertyName</code>	String	Name of the enumerated property (such as <code>genre</code>)

Output Parameters		
<code>element</code>	String	String value of the current enumerated value

Open Parameters	
<code>output</code>	Rendered once for each defined enumerated value for the specified property
<code>error</code>	Rendered if an exception occurs, the property is not an enumerated value, or no enumerated values are found

Tasks

Create the Custom Servlet Bean

1. Write an `EnumForEach` Java class and save it in the `src/dynamusic` folder of the `DynamusicEAP` module. Use the following guidelines when writing your class:

- Your class should reside in the `dynamusic` package.
- For readability, import the following:
 - `atg.servlet.*`
 - `atg.repository.*`
- Your class should extend the `atg.servlet.DynamoServlet` class.
- Your servlet bean should include a method with the following signature:

```
public void service(DynamoHttpServletRequest request,
DynamoHttpServletResponse response) throws
ServletException, IOException
```

- The `service()` method should retrieve the following `String` input parameters: `repositoryName`, `itemDescriptorName`, and `propertyName`.
- The `service()` method should perform the following:
 - Given the `repositoryName` input parameter, find the repository component object by using the servlet's `resolveName()` method.

Note: `resolveName()` is inherited from `atg.nucleus.GenericService`, which is a superclass of `DynamoServlet`. For details, see the *API Platform Reference*.

The code should look something like this:

```
repository = (Repository) resolveName(repositoryName);
```

- Use the provided utility class `dynamusic.EnumeratedProperties` (located in `DynamusicEAP/src/dynamusic`) to get an array of strings representing all the possible enumerated values for the specified property. Use the method signature below.

Note: This method throws an exception that you must catch.

```
public static String[] getEnumeratedProperties(Repository
pRepository, String pItemDescriptorName, String
pPropertyName) throws RepositoryException
```

- Loop through the enumerated values of the property and, for each one, set the `element` output parameter to the enumerated value and render the `output` `oparam`.
- If no property values are returned, or if an exception is thrown, render the `error` `oparam`.

2. Save `EnumForEach.java`, confirming that it was compiled without error.
3. Create a new globally scoped component called `/dynamusic/EnumForEach` based on `dynamusic.EnumForEach`. You do not need to configure any properties for this component.
4. Deploy your changes by using the deploy task on the Ant tab.

Use the Custom Servlet Bean in a JSP

5. Open the `config/dynamusic/songs.xml` file and find the `song.genre` property. Note the list of enumerated options. Just review this file; do not edit it for this practice.
6. Edit the `newsong.jsp` page. Find the form field where the `genre` property is set. Note that it constructs a select box by using hard-coded option tags. Replace the hard-coded option tags by using the `EnumForEach` servlet bean to generate dynamic option tags for each defined enumerated value for `genre`.

Tip: Do not forget to import that servlet bean on your page unless you put the full pathname of the servlet bean in the select tag.

Be sure to set the three input parameters required by `EnumForEach`. Use the `output` and `error` open parameters to specify the dynamic output. In the `output` `oparam`, write an HTML option tag, using the `element` output parameter both as the option value and the display string.

7. Run the `update_files` task on the Ant tab in Eclipse.
8. Load Dynamusic, navigate to the **Add Song** page (from an album details page), and verify that the drop-down list of possible song genres displays the appropriate values.

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