

**Foundations of ATG  
Application Development Rel  
10.1.2**

**Student Guide**

D81101GC11

Edition 1.1

October 2013

D84082

**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.

## Contents

### 1 Essential Skills for Building on the ATG Platform

Student Introductions	1-2
Course Objectives	1-3
Agenda	1-4
Objectives	1-5
Road Map	1-6
Oracle ATG Web Commerce	1-7
Oracle Commerce Architecture	1-8
Core Concept: Component	1-9
Core Concept: Repository	1-10
Core Concept: Profile	1-11
Core Concept: Scenario	1-12
Dynamusic.com	1-13
Instructor Demonstration: Dynamusic.com	1-14
ATG Developer Tools: Eclipse	1-15
ATG Developer Tools: ACC	1-16
ATG Developer Tools: Server Admin	1-17
Database Requirement	1-18
ATG File Organization	1-19
home directory	1-20
Quiz	1-21
Road Map	1-25
ATG Application: Overview	1-26
Creating an Application Module in Eclipse	1-27
Module Dependencies	1-29
Creating a Java EE Application	1-30
Road Map	1-31
Deploying ATG Applications	1-32
Application Assembly	1-33
ATG Documentation	1-34
Summary	1-35
Practice 1 Overview: Essential Skills for Building on the ATG Platform	1-36

## **2 Using Components**

- Objectives 2-2
- Nucleus Components 2-3
- Dynamusic and Components 2-4
- Nucleus 2-5
- JavaBean 2-6
- Nucleus Component 2-7
- Multiple Components from One Class 2-8
- Configuring Components 2-9
- Component Names 2-10
- Instructor Demonstration: Components 2-11
- Properties Files 2-12
- CONFIGPATH Environment Variable 2-13
- Example: CONFIGPATH Layering 2-15
- Setting CONFIGPATH 2-16
- Viewing Configuration Layers 2-17
- CONFIGPATH Layer Usage 2-18
- Component Instantiation 2-19
- Configured Versus Live Property Values 2-20
- Component Scope 2-21
- Global Scope 2-22
- Session Scope 2-23
- Request Scope 2-24
- Example: Scope 2-25
- A Prominent Session-Scoped Component 2-26
- Instructor Demonstration: SessionManager 2-27
- Logging 2-28
- Types of Logging Messages 2-29
- Log Files 2-30
- Console Logging 2-31
- GLOBAL.properties 2-32
- Quiz 2-33
- Summary 2-38
- For More Information 2-39
- Practice 2 Overview: Using Components 2-40

## **3 Dynamic Pages**

- Objectives 3-2
- Dynamusic.com Pages 3-3
- Road Map 3-4
- JSP: Overview 3-5

- JSP Review: Syntax 3-6
- JSP Review: Working with Objects 3-7
- JSP Review: Using Tag Libraries 3-8
- JSP Review: JSTL 3-9
- Road Map 3-11
- DSP Tag Library 3-12
- Creating and Editing JSPs 3-14
- Road Map 3-15
- Pages and Components 3-16
- Displaying Component Properties 3-17
- Importing Components 3-18
- Tag Converters 3-19
- Setting a Component Property 3-20
- Quiz 3-21
- Practice 3 Overview: Dynamic Pages: Part 1 3-22
- Road Map 3-23
- Including Page Fragments 3-24
- Example: Including Page Fragments 3-26
- Page Parameters 3-28
- Accessing Parameters 3-29
- Setting Parameters 3-30
- Passing Parameters 3-31
- Example: Passing Parameters 3-32
- Dynamic Parameter Values 3-35
- Passing Parameters Via Links 3-36
- Quiz 3-37
- Road Map 3-39
- What Is a Dynamo Servlet Bean? 3-40
- DSB: Simple Example 3-41
- ProtocolChange Servlet Bean: Example 3-42
- ProtocolChange Servlet Bean: Example – Oparam 3-43
- ProtocolChange Servlet Bean: Example – Input and Output Parameters 3-44
- ForEach Servlet Bean: Example 3-45
- Using Dynamo Servlet Beans in a Page 3-46
- Servlet Bean Parameters: ForEach 3-47
- Example: ForEach Servlet Bean 3-54
- Using Servlet Bean Snippets 3-55
- Some Frequently Used Servlet Beans 3-56
- Quiz 3-57
- Road Map 3-59
- Nested DSP Tag Restrictions 3-60

Using dsp:valueof with EL Variables	3-62
Summary	3-63
For More Information	3-64
Practice 3 Overview: Dynamic Pages: Part 2	3-65

## 4 Using ATG Repositories

Objectives	4-2
Road Map	4-3
The Problem	4-4
The Solution	4-5
ATG Repositories	4-6
Where Repositories Are Used	4-7
Road Map	4-8
Dynamusic.com Repositories	4-9
Repository Items	4-10
Item Relationships	4-12
Demonstration: Repositories and the ACC	4-13
Road Map	4-14
Using Repositories with Dynamic Pages	4-15
Repository Servlet Beans	4-16
Using an Item Lookup Droplet	4-17
Quiz	4-19
Road Map	4-21
Repository Query Servlet Beans	4-22
Using RQL Servlet Beans	4-23
Repository Query Language (RQL)	4-24
RQL Statements	4-25
RQL Query Conditions	4-26
RQL Queries with Item References	4-27
RQL Collection Queries	4-28
Parameterized RQL Queries	4-29
RQLQueryRange: Example	4-30
RQL Query and Item Lookup Together	4-32
Quiz	4-34
Summary	4-36
Additional Information	4-37
Practice 4 Overview: Using Repositories	4-38

## 5 Configuring SQL Repositories

- Objectives 5-2
- Object to Relational Mapping 5-3
- Item Descriptor 5-4
- Repository Access to Database 5-5
- Road Map 5-6
  - Step 1: Set Up Database Tables 5-7
  - Step 2: Configure Database Access 5-8
  - Step 3: Create a SQL Repository Component 5-9
- Quiz 5-10
- Road Map 5-11
- Repository Definition File 5-12
- XML Syntax Review 5-13
- Structure of <gsa-template> 5-14
- Design Objects, Not Tables 5-16
- Dynamusic SongsRepository Object Model 5-17
- Item Descriptors 5-18
  - <item-descriptor> Tag 5-19
  - <table> Tag 5-20
  - <property> Tag 5-21
- Other Types of Properties 5-22
  - Enumerated Properties 5-23
  - Multivalued Properties 5-24
  - Multivalued Properties: Definition File 5-25
  - Item Properties 5-26
  - Item Properties: Definition File 5-27
  - Cross-repository Item Properties 5-28
  - Multi-item Properties: One-to-Many 5-29
  - Multi-item Properties: One-to-Many in the Definition File 5-31
  - Types of Multivalue Properties 5-32
  - Quiz 5-33
  - Item Descriptor Inheritance 5-35
  - Dynamusic Artist Subtypes 5-36
  - Subtypes in the Database: Example 5-37
  - Base Item Descriptor: artist 5-38
  - Subtype Item Descriptor: artistType 5-40
  - Advantages of Subtyping 5-42
  - Creating a Subtype Item 5-43
  - ID Generation 5-44
  - Grouping Properties 5-45
  - Example: Grouping Properties 5-46

Cascading Relationships	5-47
Display Property Attribute	5-48
Operational Tags	5-49
Repository Definition Summary	5-50
Quiz	5-52
Road Map	5-54
Setting Up a Test Data Source	5-55
Testing a SQL Repository	5-56
Repositories and the ACC	5-57
Editing XML	5-58
SQL Repository Development Checklist	5-59
Summary	5-60
Additional Information	5-61
Practice 5 Overview: Configuring SQL Repositories: Practice 5-1	5-62
Practice 5 Overview: Configuring SQL Repositories: Practice 5-2	5-63

## **6 Using Repository Form Handlers**

Objectives	6-2
Dynamusic Forms	6-3
ATG Forms	6-4
ATG Form Handlers	6-5
Road Map	6-6
RepositoryFormHandler	6-7
Configuring a Repository Form Handler	6-8
Handling Dynamic Properties	6-9
Accessing Dynamic Bean Properties	6-10
Using DSP Form Tags	6-11
Using DSP Form Tags in the JSP	6-12
DSP Input Tags	6-13
Default Values	6-14
DSP Input Tag Options	6-15
Multivalue Form Input	6-16
Multivalue Form Input in the JSP	6-17
HTML Hidden Input Fields	6-18
DSP Hidden Input Fields	6-19
Quiz	6-20
Submission Handler Methods	6-21
Submission Handler Methods in the JSP	6-22
Page Redirection	6-23
Page Redirection Properties	6-24
Setting Redirect Properties	6-25

Common Form Page Design	6-26
A Common Form Page Design: JSP Expression	6-27
Quiz	6-28
Error Handling	6-29
Error Handling: ErrorMessageForEach	6-30
ErrorMessageForEach: Example	6-31
Updating Existing Repository Items	6-32
Request-Scope Form Handlers	6-33
Populating a RepositoryFormHandler	6-40
Setting the Item ID in the Form Handler	6-41
Passing the Item ID to the Action Page	6-42
Passing the Item ID to a Redirect Page: dsp:getvalueof	6-43
Passing the Item ID to a Redirect Page: Format droplet	6-44
Advanced Features	6-45
Quiz	6-46
Road Map	6-47
Search Form Handlers	6-48
Using SearchFormHandler	6-49
Configuring a Keyword Search Component	6-50
Using SearchFormHandler	6-51
Using SearchFormHandler: Results	6-52
Search	6-53
Quiz	6-54
Summary	6-55
Additional Information	6-56
Practice 6 Overview: Using Repository Form Handlers	6-57

## 7 Extending the Profile Repository

Objectives	7-2
User-Centered Application Design	7-3
Profile Properties	7-4
Profile Properties: Example	7-5
Plan for Learning	7-6
Plan for Learning: Scenario	7-7
Reasons to Extend the Profile for Dynamusic	7-8
Dynamusic Profile Extensions: Example	7-9
Road Map	7-10
Profile Repository: Overview	7-11
ProfileAdapterRepository Component	7-12
Profile Component	7-13
Internal and External Users	7-14

Road Map	7-15
Profile Repository Layers	7-16
Base Layer Profile Repository Items	7-17
Modifying the Profile Repository	7-18
Extending the Profile Repository	7-19
Quiz	7-20
Road Map	7-21
Adding a Simple Property: subscriber	7-22
Step 1: Add an Auxiliary Table	7-23
Step 2: Create Repository Definition Layer – XML file	7-25
Step 2: Create Repository Definition Layer – user Item Descriptor	7-26
Road Map	7-27
Example: Add a Business Object to the Profile	7-28
Step 1: Add Necessary Tables	7-29
Step 2: Add viewedSongs to the user Item	7-30
Removing Properties via Layering	7-31
Transient Properties	7-32
Viewing Combined Layers	7-33
Instructor Demonstration: Combined XML File	7-34
Property Attributes	7-35
Road Map	7-36
Dynamusic Profile Forms	7-37
ProfileFormHandler	7-38
Accessing Profile Attributes in a Form	7-39
Profile Submission Handler Methods	7-40
Profile Page Redirection	7-41
What the login Handler Does	7-42
ProfileFormHandler Properties	7-43
Quiz	7-44
Summary	7-45
For More Information	7-46
Practice 7 Overview: Extending the Profile Repository	7-47

# 1

## Essential Skills for Building on the ATG Platform

ORACLE®

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

# Student Introductions

Please introduce yourself to the class by answering these questions:

- What is your name?
- Where do you work? What is your job?
- Have you worked with ATG already? If so, what have you done with it?
- How much experience do you have with Java and object-oriented design and development?
- How will your company be using ATG?
- Why are you taking this course?



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

## Course Objectives

After completing this course, you should be able to:

- Implement JavaServer Pages that use ATG services
- Configure and access SQL databases using Repositories
- Process forms using the out-of-the-box form handlers
- Define customized user profile templates



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

# Agenda

Session	Lesson
<b>Day One</b>	Lesson 1: Essential Skills for Building on the ATG Platform Lesson 2: Using Components Lesson 3: Dynamic Pages
<b>Day Two</b>	Lesson 4: Using ATG Repositories Lesson 5: Configuring SQL Repositories
<b>Day Three</b>	Lesson 6: Using Repository Form Handlers Lesson 7: Extending the Profile Repository



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

# Objectives

After completing this lesson, you should be able to:

- Discuss the products in the ATG Commerce product suite
- Use ATG Developer Tools
- Create ATG application modules
- Deploy and run an ATG application

# Road Map

- Getting started
- Creating an ATG application
- Deploying and running ATG applications



ORACLE

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

## Oracle ATG Web Commerce

- ATG is
  - A platform for developing dynamic web applications
  - A set of classes and tools
  - An engine for components
- ATG is used by some of the largest companies and sites in the world.
- The platform is fast, scalable, and based on Java EE standards.



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

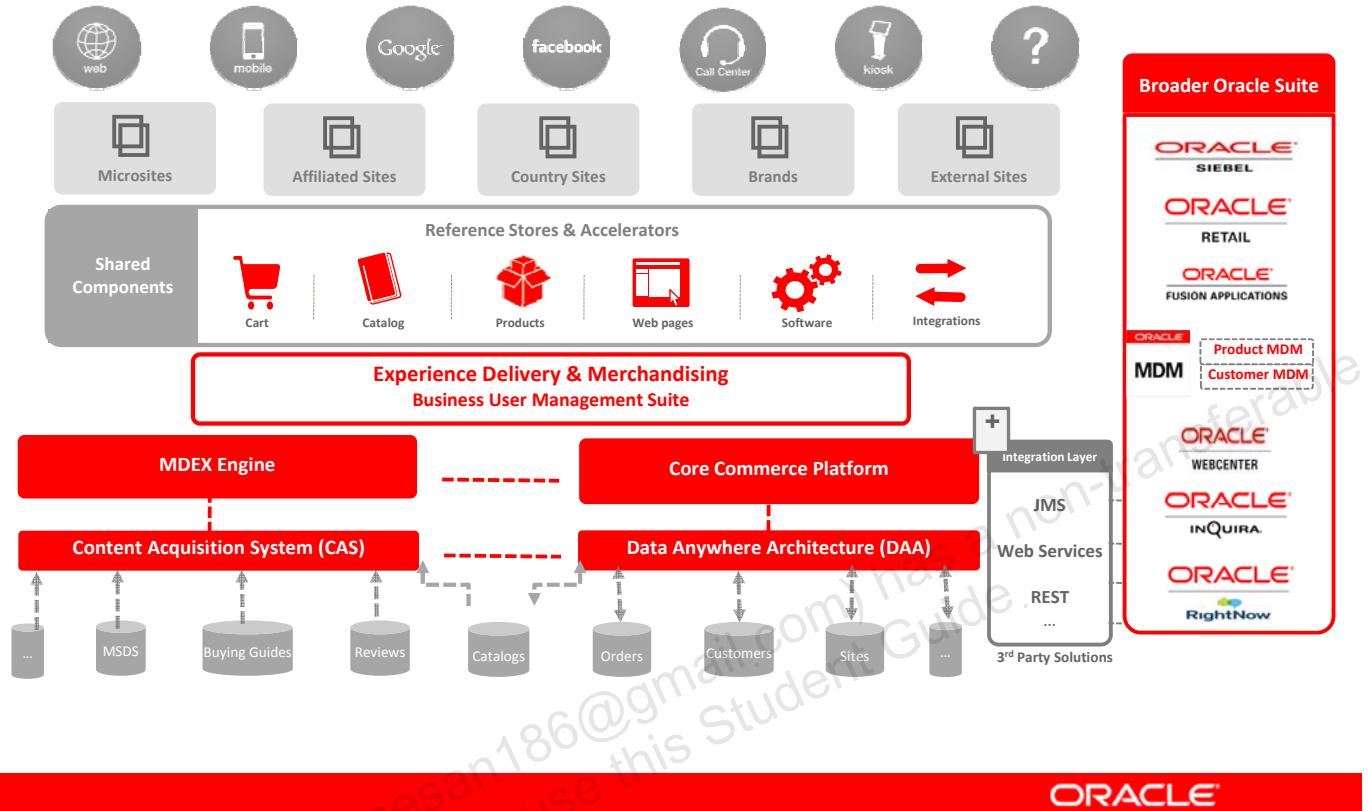
The ATG Application Framework supplies numerous services and functions implemented on top of the standard Java EE platform, saving developers the expense of finding such services or writing those services themselves. It also enables a development methodology that is uniform across the entire platform.

Industry standards on which ATG products are built include:

- **Java:** A programming language that does not force platform dependence
- **JavaBeans:** A Java specification for developing reusable components
- **Servlets:** Java programs that run on the server, as opposed to the client
- **JavaServer Pages (JSPs):** Part of the Java EE specification; a means to combine functionality and presentation in web pages
- **Wireless Application Protocol (WAP/WML):** The ability to handle web requests from WAP-enabled devices such as mobile phones

For a more complete list of the industry standards in ATG, please refer to the “Overview” section in the ATG Documentation Set.

# Oracle Commerce Architecture



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

ORACLE

The two product lines that comprise Oracle Commerce are Oracle Endeca Commerce and Oracle ATG Web Commerce. Oracle Endeca Commerce focuses on Experience Management including search and guided navigation of a website. The Oracle ATG Web Commerce platform handles catalog management, order management, shopping cart services, and multisite and merchandising activities.

Oracle Master Data Management is a comprehensive platform that delivers consolidated, consistent, and authoritative master data across the enterprise and distributes this master information to all operational and analytical applications. Its capabilities are designed for mastering data across multiple domains ranging from Customer, Supplier, Site, Account, Asset, Product, and many others.

## Core Concept: Component

- A Nucleus component is a Java class that is the basic building block for all processing of information that flows between users and a website.
- Components perform functions such as:
  - Data storage and retrieval
  - Personalization with business rules
  - E-Commerce business logic



**Java Component**

**ORACLE**

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

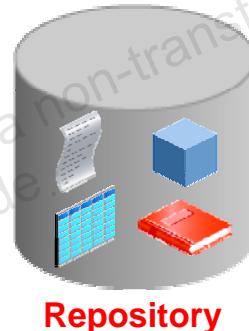
A Java component can perform a very simple function or something more complex. Profiles, targeters, and content repositories are all examples of Java components.

A key element of ATG is the separation of programming and presentation. Dynamic pages use components, but do not need to have code built into them. The programming functionality is encapsulated within components. This means that Java programmers do not need to create the web pages. Instead, your HTML designers can create pages and access dynamic functionality in the page by simply referencing properties from the components. Thus Java programmers can focus their talents on building useful components, whereas HTML designers can focus on building great looking and functional pages.

ATG comes with many out-of-the-box components that you can configure for your site. You can also create your own custom components.

## Core Concept: Repository

- A *repository* is the basic method of **data access** in ATG. It may manage:
  - Structured (relational) data
  - Documents
  - Multimedia data
- Examples of repositories include:
  - Profile repository
  - Content repositories
  - Commerce repositories



ORACLE

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

ATG products use many different repositories. Some contain relational data such as the profile repository. Other types of repositories may contain or reference documents such as a content repository.

ATG repositories provide a unified interface to a variety of external data sources.

Repositories may be implemented using RDBMS, LDAP directories, other applications, or specialized Java Components.

## Core Concept: Profile

A *profile* is information about a particular person visiting a website. The information may include:

- Name, address, and so on
- System environment
- Browsing behavior
- Application-specific attributes



**Profile**

**ORACLE**

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

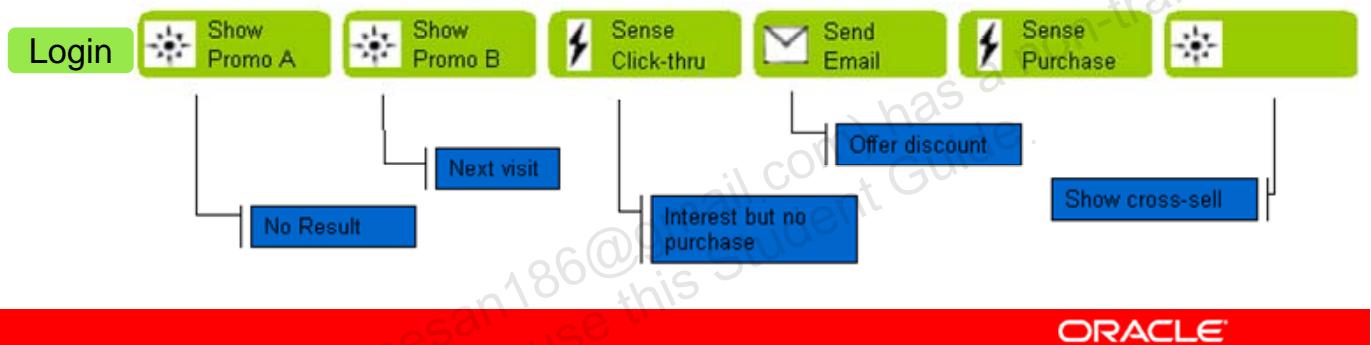
Each user profile contains information about a visitor to your site. A profile is simply a collection of *attributes* that describe the user. You determine what information you want to track about your users. Typical attributes are the user's name, address, phone number, and so on. The values for the attributes come either through explicit information provided by the user in a form or through implicit information such as the user's system environment (for example, which browser is being used) and browsing behavior.

Browsing behavior refers to the pages on the site the customer visits. For instance, you may track whether a customer spends more time in the sports section of your site or the movie section. This leads to the concept of specially coded attributes. Each site has its own important attributes to track about visitors. The profile implementation in ATG makes it easy for you to customize your site by adding the attributes important to you.

## Core Concept: Scenario

A scenario is a *game plan* that associates sequences of events with actions. After it is triggered, actions may target specific individuals, groups of users, or all users. Actions may include:

- Delivering personalized content
- Sending email
- Displaying promotional information



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

An important benefit of scenarios is that their actions can happen over time. They are not necessarily linked to a user's single visit to your site. For example, you are running a marketing seminar and want people to register. You track whether the person has registered in the profile. When a visitor, who has not yet registered, comes to your site, you want to advertise the seminar. You show an image with a link to information and the registration form. If the customer clicks the link and registers, you send a confirmation email. If the user does not register, you send an email encouraging registration. If the user does not register within two weeks of the first visit, you send a second email advertising your seminar.

## Dynamusic.com

- Dynamusic is a fictional company that allows users to download MP3s.
  - Revenue comes from monthly subscription fees, referral fees for album purchases from a retail site, and advertising for concerts and venues.
- The practices for the course develop a subset of Dynamusic's functionality.

 ORACLE

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

Throughout this course (in both the practices and in-class presentations), you will see examples drawn from a fictional site called **Dynamusic.com**.

To understand the examples drawn from Dynamusic, it is helpful to understand Dynamusic's business model.

# Instructor Demonstration: Dynamusic.com

Home

Welcome, Arthur!

This week's featured songs are:

- Almost Green
- Where Did I Put My Raincoat?
- Spoons of Summer
- Whatever

Featured Album:

Women & Men - Autumn Winters

ORACLE

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

Dynamusic features include the ability to:

- Browse artists, albums, and songs
- Search for songs based on title or description
- Browse and search for concerts and venues
- Read and write album, song, and concert reviews

There are different user types:

- Nonpaying users can browse but not download songs, except “amateur” songs uploaded by the artist.
- Registered users get additional functionality based on user knowledge, such as links to venues in their state.
- Fee-paying users get access to all site functionality.

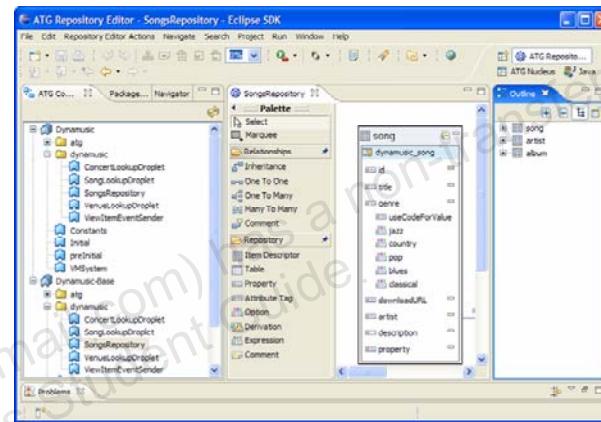
Users can do the following:

- Select preferred genres (jazz, classical, and so on), which can be used for song recommendations, as well as for finding other users with similar interests.
- Opt to share their profile information, including a “description” of themselves, with other users; Dynamusic will forward email preserving user privacy.
- Create “playlists” of their favorite songs and optionally share those with other users.

Registered users who are amateur musicians can upload songs to share with interested users.

## ATG Developer Tools: *Eclipse*

- Eclipse is an open-source development environment.
  - Very extensible through plug-ins
- ATG provides powerful Eclipse plug-ins to:
  - Create and assemble ATG applications
  - Browse and edit Nucleus Components
  - Assist in editing JSPs
  - Edit repository definitions



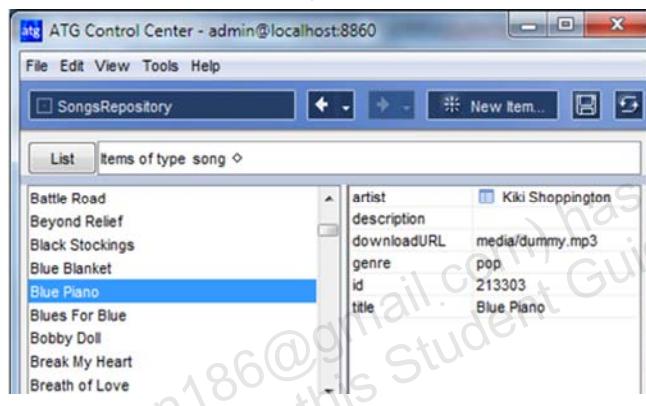
ORACLE

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

Eclipse is a very powerful development environment, which is mostly beyond the scope of this course. Here you will only learn the bare minimum of Eclipse required to complete the practices. For more information about Eclipse, see the website <http://www.eclipse.org/>, or refer to any number of discussion forums, books, and other resources on the Internet.

## ATG Developer Tools: ACC

- The ATG Control Center (ACC) is a GUI tool that helps developers configure and personalize website content.
- Use the ACC to:
  - Browse and edit component configurations and live values
  - Build scenarios
  - View and edit repository data



ORACLE

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

The ACC may be run in one of three modes:

- **Same VM:** The ACC application runs in the same Java Virtual Machine (JVM) as your ATG application. This can improve performance by requiring only a single JVM to run on your computer but may slow the application itself. You start the ACC in the same JVM via the Dynamo Administration user interface (UI) as discussed on the next slide.
- **Different VM on the same computer:** The ACC will run from the same installation as your ATG application but in a separate process (also started via the Dynamo Administration UI).
- **Different computer:** You can run the ACC as a stand-alone application that connects over a network to an ATG server instance running on a different machine. The stand-alone ACC is a separate application that must be installed on the client.

## ATG Developer Tools: Server Admin

- The ATG Dynamo Administration UI (Admin UI) lets you manage each ATG instance directly.
- Use the Admin UI to view and change repository data, view live components, and browse session data.

`http://localhost:7103/dyn/admin`

The screenshot shows the ATG Dynamo Administration User Interface. On the left, there is a login dialog box titled "Enter Network Password". It has fields for "Site" (localhost), "Realm" (Dynamo Admin Server), "User Name" (highlighted in a red box), and "Password" (highlighted in a red box). A checkbox for saving the password is also present. On the right, the main dashboard is titled "Dynamo Administration" and displays the version "Version ATGPlatform/10.0". It features several links: "Configuration Manager" (Change the configuration properties of your application), "Component Browser" (Browse the hierarchy of components registered), "Admin ACC" (Start the ATG Control Center for this Dynamic Application), "Change Password" (Change the passwords for ATG Dynamo accounts), and "JDBC Browser" (Browse a database through a JDBC connection, create and drop tables, and execute database queries).

ORACLE

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

The ATG Dynamo Administration User Interface provides convenient tools to administer your ATG products.

To run the Admin UI, you must include the `DafEar.admin` module in your assembled application. Application assembly will be discussed later in the lesson.

The URL to access the Admin UI varies depending on the configuration of your application server and/or web server. In this course, you will be using WebLogic, using the HTTP port 7103. Use username **admin** and password **password1**.

Note that ATG must be running for you to access the Admin UI.

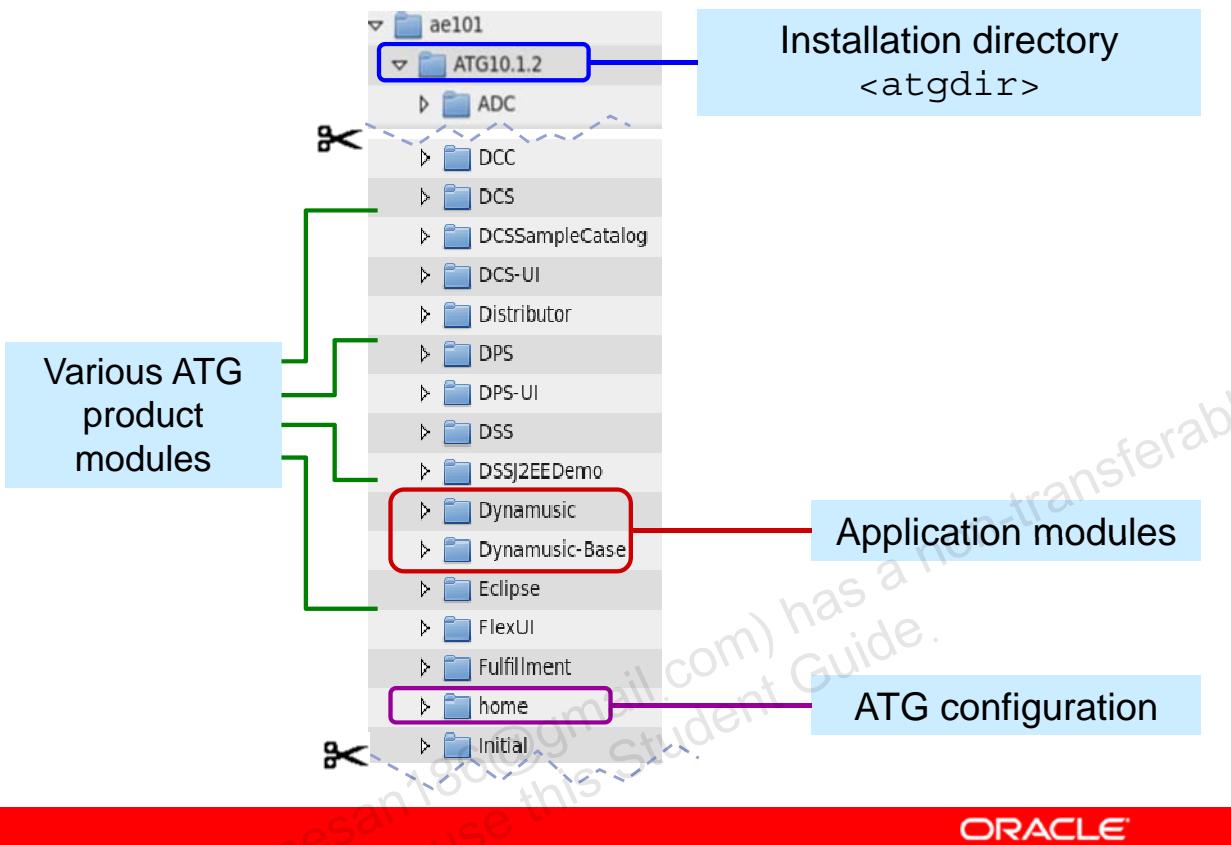
## Database Requirement

- ATG requires a SQL database to run.
- Practices for this course use Oracle Database Express Edition.
- Database connection configuration is covered in the lesson titled “Configuring SQL Repositories.”



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

# ATG File Organization



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

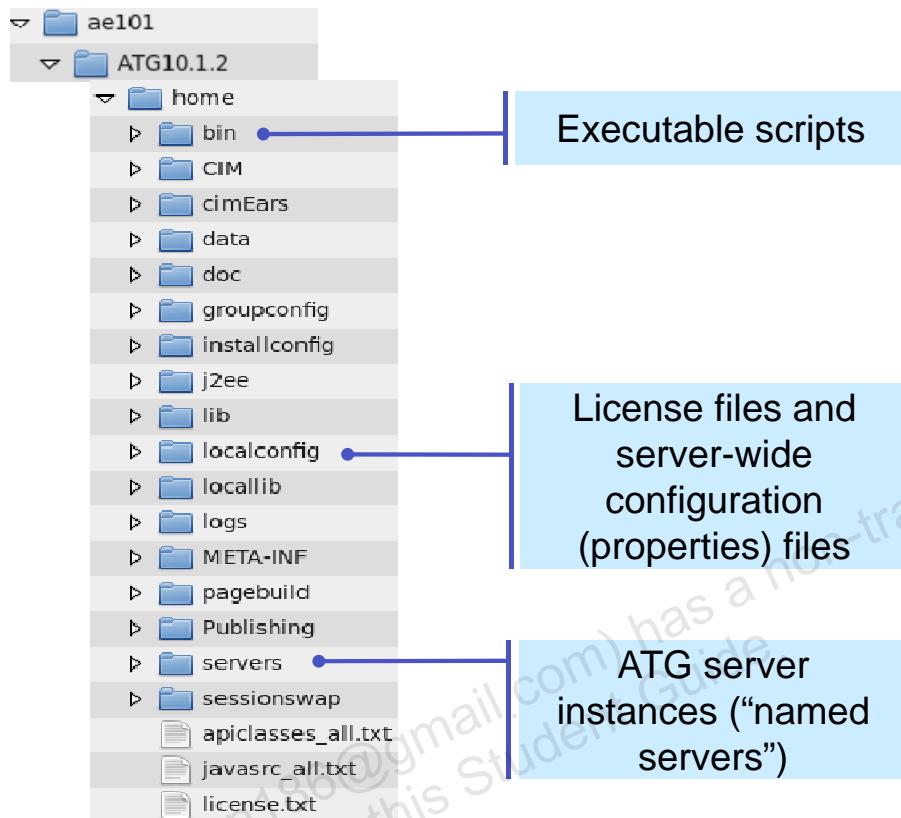
As an ATG developer it is important for you to become familiar with the general layout of the ATG installation directory.

Throughout this course and the documentation, this directory is referred to as `atgdir`. The practices for this course use the following location `/ae101/ATGx.x`.

The top-level directory contains mostly modules, including:

- ATG product modules (DAS, DPS, and so on)
- Sample application modules
- ATG server-wide configuration directory (home)
- Your application modules

## home directory



**ORACLE**

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

The ATG home directory (often referred to as `DYNAMO_HOME`), located in the `<atgdir>` directory, contains scripts, server-wide configuration, and server settings. (The term “Dynamo” is a holdover from ATG’s original product line, called Dynamo. Although ATG products no longer use the Dynamo name, you will see frequent references to “Dynamo” throughout the product and documentation. It is often used as a synonym for the ATG Platform.)

The `localconfig` directory contains properties files that are used to define and configure system-wide components. (There are actually a variety of places where properties files are located; this directory contains properties files that set deployment-level configuration parameters for an entire cluster, rather than application-specific settings [which are in application modules] or server-specific settings [which are in the `servers` directory].)

The `servers` directory contains ATG server instances (also called “named servers”) that are used to separate core functions of ATG instances in a cluster.

The `bin` directory contains all executables that come with ATG.

`locallib` becomes part of CLASSPATH when the server starts.

# Quiz

The basic building block for all processing of information that flows between users and the website is:

- a. Profile
- b. Scenario
- c. Nucleus component
- d. Repository

 ORACLE

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

**Answer: c**

# Quiz

The basic method of data access in ATG is:

- a. Profile
- b. Scenario
- c. Nucleus component
- d. Repository



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

**Answer: d**

# Quiz

A \_\_\_\_\_ is a game plan that associates sequences of events with actions.

- a. Profile
- b. Scenario
- c. Nucleus component
- d. Repository



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

**Answer: b**

# Quiz

A \_\_\_\_\_ is information about a particular person visiting a website.

- a. Profile
- b. Scenario
- c. Nucleus component
- d. Repository



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

**Answer:** a

# Road Map

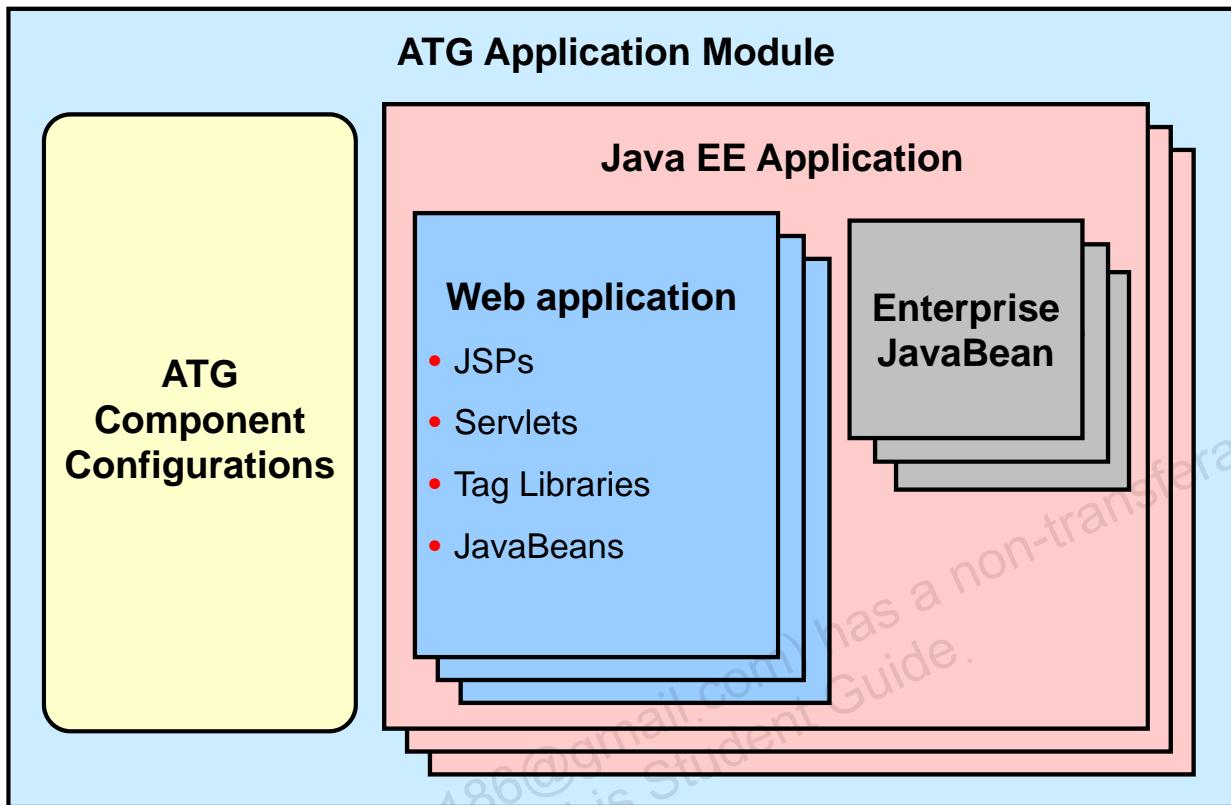
- Getting started
- Creating an ATG application
- Deploying and running ATG applications



ORACLE

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

# ATG Application: Overview



ORACLE

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

An ATG application has several “levels”:

1. An application “module” (folder) that provides a container for your entire application
2. A folder containing configuration files for Nucleus components
3. A Java EE application inside the module, called an “Enterprise Application,” which may contain one or more “Web Applications” (the user-facing front end of an application), as well as JavaBeans (back-end components), and perhaps client applications (applications downloaded via a client’s web browser)
4. The web application contains all the elements of the front end of the application, including JSPs, servlets, and so on

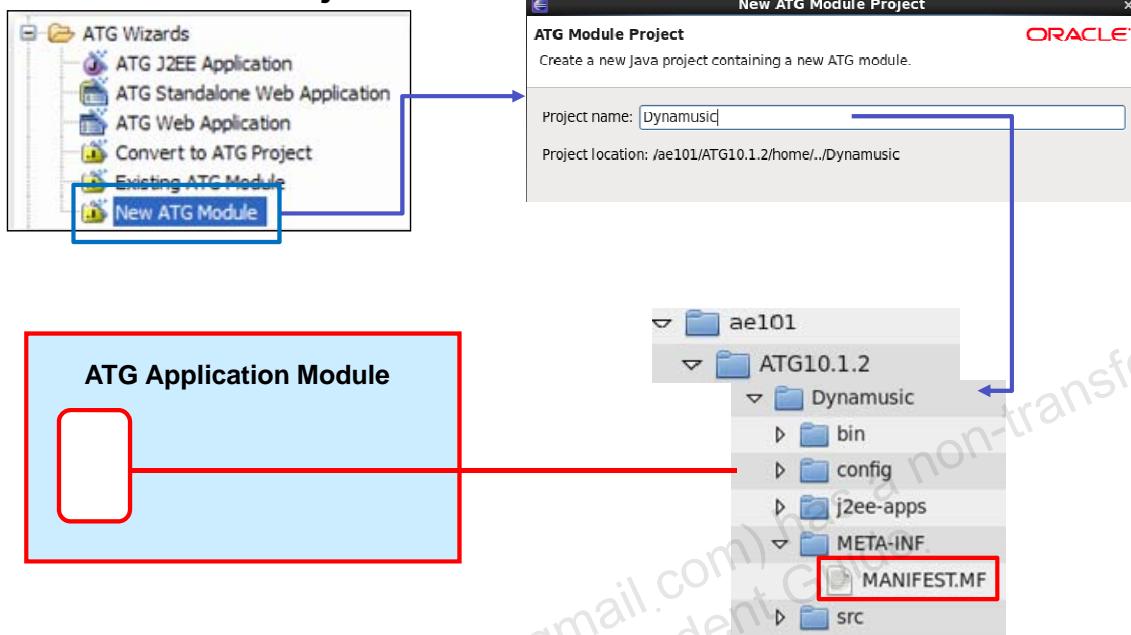
The course focuses primarily on the web application level, because that is the context in which JSPs run.

**Java Enterprise Edition (Java EE)** is a standard platform for writing server-side applications in Java. It determines the way server-side applications should be written and packaged in order to be portable across Java EE implementations, and also mandates what services a Java EE application server must provide to the applications it hosts.

For more information about Java EE, see <http://www.oracle.com/technetwork/java/index.html>, which includes documentation, API references, tutorials, articles, and other useful resources. Although ATG supports Java EE, it is beyond the scope of this course.

# Creating an Application Module in Eclipse

File > New > Project



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

ORACLE

ATG modules, and the Java EE applications contained within, must follow a certain folder structure, and contain declaration files that follow a specific format. To create these files and folders, most developers prefer to use tools such as the ATG Eclipse plug-in wizards, or the ATG Control Center (ACC).

This slide shows the Eclipse **New ATG Module** wizard (accessed from the File > New > Project menu) The wizard prompts for a project name, which will become the name of your module—in this example, `Dynamusic`. You also supply the root directory, `<atgdir>` (where you installed ATG).

Modules may be nested. For example, the ATG Commerce Reference Store module (`CommerceReferenceStore`) contains nested module folders such as `Estore` (under `CommerceReferenceStore/Store`) and `Storefront` (also under `CommerceReferenceStore/Store`). These modules are referred to, respectively, as `Store.Estore` and `Store.Storefront`. The thing to note is that references to nested ATG *product* modules, such as those just mentioned, sometimes do not include the top-level folder. For custom modules, the top-level folder is generally included in the reference. Also note that starting a “top-level” module does not automatically start all nested modules. Each module must be referenced individually in the usual way.

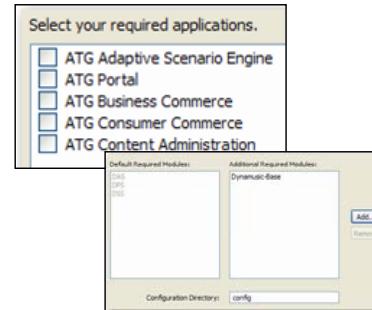
Every ATG application module must have a MANIFEST.MF file, contained in a META-INF directory. This is a text file that declares the location and purpose of the modules contents. The file consists of attribute-value pairs such as:

Manifest-Version: 1.0

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

## Module Dependencies

- ATG applications often depend on other modules that are started automatically when your module is started.
- Dependencies may be declared in the module manifest file.



<module-name>/META-INF/MANIFEST.MF

```
Manifest-Version: 1.0
```

```
ATG-Required: DSS
```

ORACLE

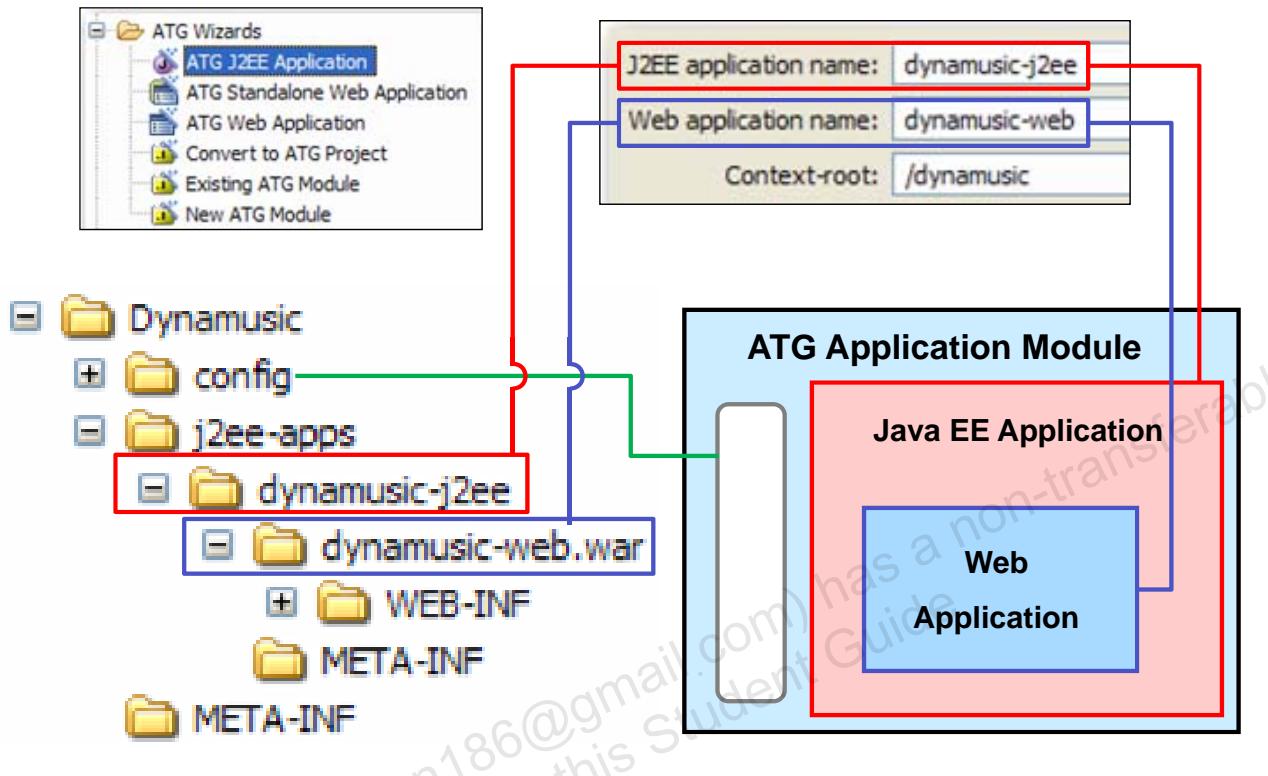
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

As shown in the example in the slide, the Eclipse ATG wizards will prompt for dependencies—both ATG product modules and your own application modules—and will automatically include those you select in the Manifest.

**ATG-Required:** Typically, this list includes DSS at a minimum. If your application uses facilities provided by other application layers (such as commerce functionality from DCS), you need to include those modules in the list as well. (You need only include the top-level module you require; additional modules required by the modules you require are automatically started. For example, if your application requires DSS, which requires DPS, you need to specify only DSS, not DPS.)

The complete syntax for the manifest file is described in the *ATG Programming Guide*, in the section titled “Working with Application Modules.”

# Creating a Java EE Application



ORACLE

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

The context root:

- Is the URL “mounting point” of the web application
- Allows the application developer to decide what URL site visitors enter to get to the site, such as `http://www.mycom.com/dynamusic/artists.jsp`
- Is set in the `application.xml` file

# Road Map

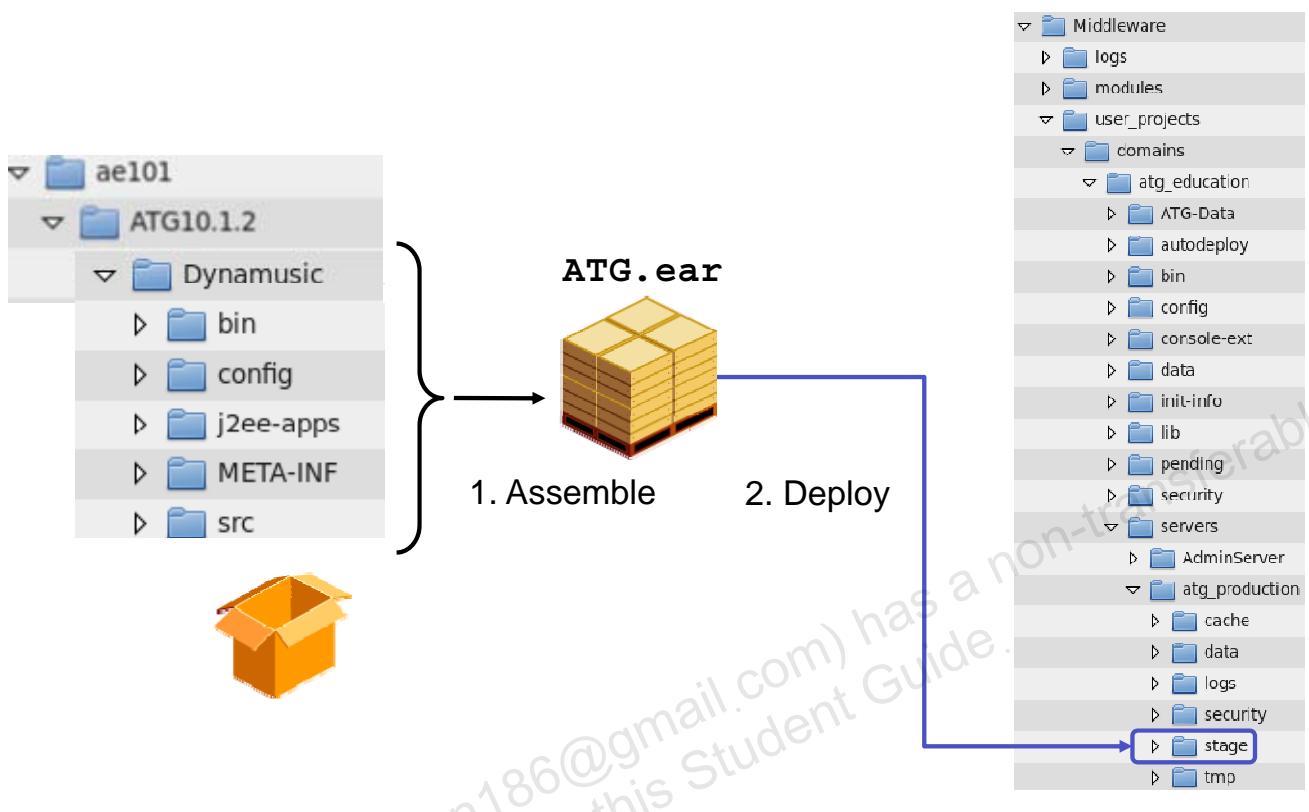
- Getting started
- Creating an ATG application
- Deploying and running ATG applications



ORACLE

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

# Deploying ATG Applications



ORACLE

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

Steps for deployment depend on which application server you are using. This course uses WebLogic.

- The process of deploying an ATG application on a WebLogic server involves:
  - Assembling the application module into an EAR (Enterprise Application Archive)
  - Deploying that ear to WebLogic. This can either be done through the WebLogic Administration Console or you can script deployment in various ways. The practices for this course use an Ant task to deploy the application.
- WebLogic deploys the application to the server's stage directory. In this course, the server is named atg\_production.

# Application Assembly

Four choices for assembly:

- Command line: <atgdir>/home/bin/runAssembler
- Ant task: CreateUnpackedEarTask
- Configuration and Installation Manager (CIM)
- Wizard: ATG plug-in for Eclipse



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

Depending on your needs and your role in application management, there are several different methods for creating an EAR for your application.

- **runAssembler:** This script creates an EAR file from one or more ATG application modules. You can use this script to create a stand-alone EAR by using the –standalone flag. For more information, see the “Developing and Assembling Nucleus-based Applications” chapter of the *ATG Programming Guide* and the section on clustering on your application server of the “Configuring for Production” chapter of the *ATG Installation and Configuration Guide*.
- **Ant Tasks:** Ant is an automated Java build tool ([see http://ant.apache.org/](http://ant.apache.org/)). ATG provides two task classes for use with Ant: `CreateUnpackedEarTask` (which creates an unpacked [open directory] EAR) and `PackEarFileTask` (which packs an unpacked EAR). For more information, see the “Developing and Assembling Nucleus-based Applications” chapter of the *ATG Programming Guide*.
- **CIM** is primarily used for installing and configuring ATG products. See the *ATG Installation and Configuration Guide* for more details.
- **Eclipse plug-in:** ATG provides a plug-in for Eclipse that includes many useful tools for the ATG developer. The Application Assembly wizard provides a graphical user interface to invoke the assembler.

## ATG Documentation

The ATG documentation set includes several manuals. Most useful to developers are:

- *Programming Guide*: A detailed description of ATG programming concepts for developers
- *Page Developer's Guide*: A guide for page developers on creating JSPs within an ATG application
- *Repository Guide*: A developer's guide to the Data Anywhere Architecture
- *API Reference*: Javadoc descriptions of the ATG classes



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

## Summary

In this lesson, you should have learned how to:

- Describe the ATG product suite
- Use ATG Developer Tools
- Create ATG application modules
- Deploy and run an ATG application

# Practice 1 Overview: Essential Skills for Building on the ATG Platform

This practice covers the following topics:

- Creating a new Dynamusic module and Java EE application using Eclipse
- Assembling an application and deploying it on WebLogic



ORACLE

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

## Using Components

ORACLE®

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

# Objectives

After completing this lesson, you should be able to:

- Describe what Nucleus does
- Create and configure Nucleus components
- Work with CONFIGPATH
- Consider implications of component scope
- Use component logging properties

## Nucleus Components

Nucleus components implement all logic in the ATG framework.



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

An ATG web application consists of components that “do the work” of the application. Many components provide the business logic for the application. Some components serve to access data and others to present information.

## Dynamusic and Components

All of the “back end” at Dynamusic.com is implemented in the form of components, including components to:

- Handle access to the database where user, music, and event information is stored
- Handle user input, such as profile data and album reviews
- Apply business logic to deliver personalized content such as concert recommendations



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

Most of Dynamusic’s functionality is built from components that are included out of the box with the ATG Platform, requiring at most some custom configuration. This lesson focuses on that aspect. To learn how to create custom components for specialized functionality not included in the ATG product, take the next course: *Extending the ATG Platform*.

ATG provides developers with a framework for using and developing components independently of the “front end” of the web site—the pages that allow the user access to a site’s functionality.

# Nucleus

Nucleus is the ATG container for components.

- Provides a hierarchical name space
  - Each component has a unique full name so that pages and other components can reference it.
- Creates and initializes component instances on request
- Decouples code from configuration (configuration through text files)
- Manages component scope



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

Nucleus features:

- **Access to naming services: Path name to object mapping.** From the full path name of the component, Nucleus locates the properties file for the component and through that, the class of the component. This enables components to be connected at development time or run time.
- **Automatic instantiation.** You do not have to write the code to instantiate the component. Rather, the component is automatically instantiated by Nucleus when it is needed by the application.
- **Decoupling code and configuration.** Components can be configured at a system level or an application level without any programming, using text files. This provides a lot of flexibility and less QA necessary in development. This enables you to develop components without writing the code that assigns property values and links components, thereby pushing the configuration and assembly of the application to a separate level.

You can think of the Nucleus as a name space manager, or a component broker.

# JavaBean

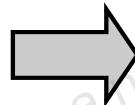
A JavaBean is a Java class with:

- A no-argument constructor
- Properties implemented by **get** and **set** methods

methods:

```
public String getItemType () ...
public void setItemType (String) ...
```

property:  
**itemType**



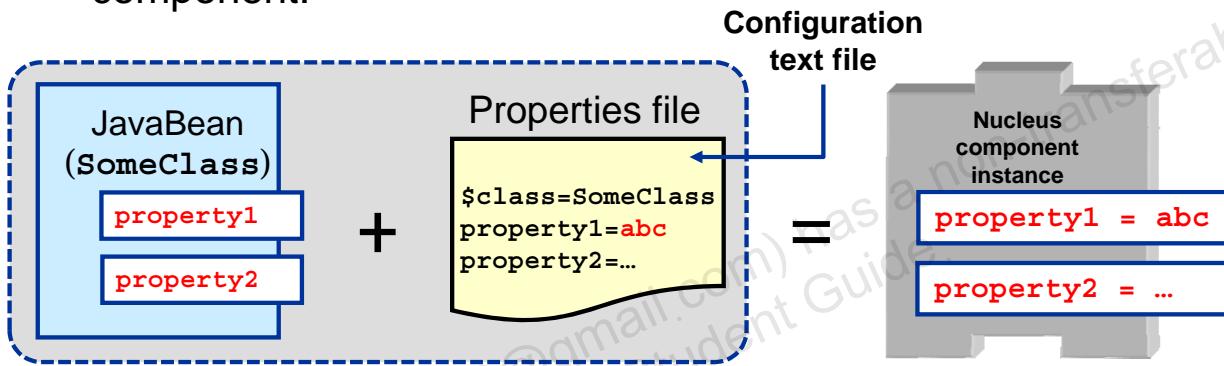
**ORACLE**

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

JavaBeans are part of the Java platform. For more information, see <http://www.oracle.com/technetwork/java/index.html>. The full JavaBean specification is not covered as part of this course. In the example in the slide, you see two methods: `getItemType` and `setItem`. These methods relate to the `itemType` property of the JavaBean.

# Nucleus Component

- A Nucleus component instance is a configured Java object.
- A .properties file is used to configure instances of a standard Java bean.
- Nucleus sets configured values on each new instance of a component.



**ORACLE**

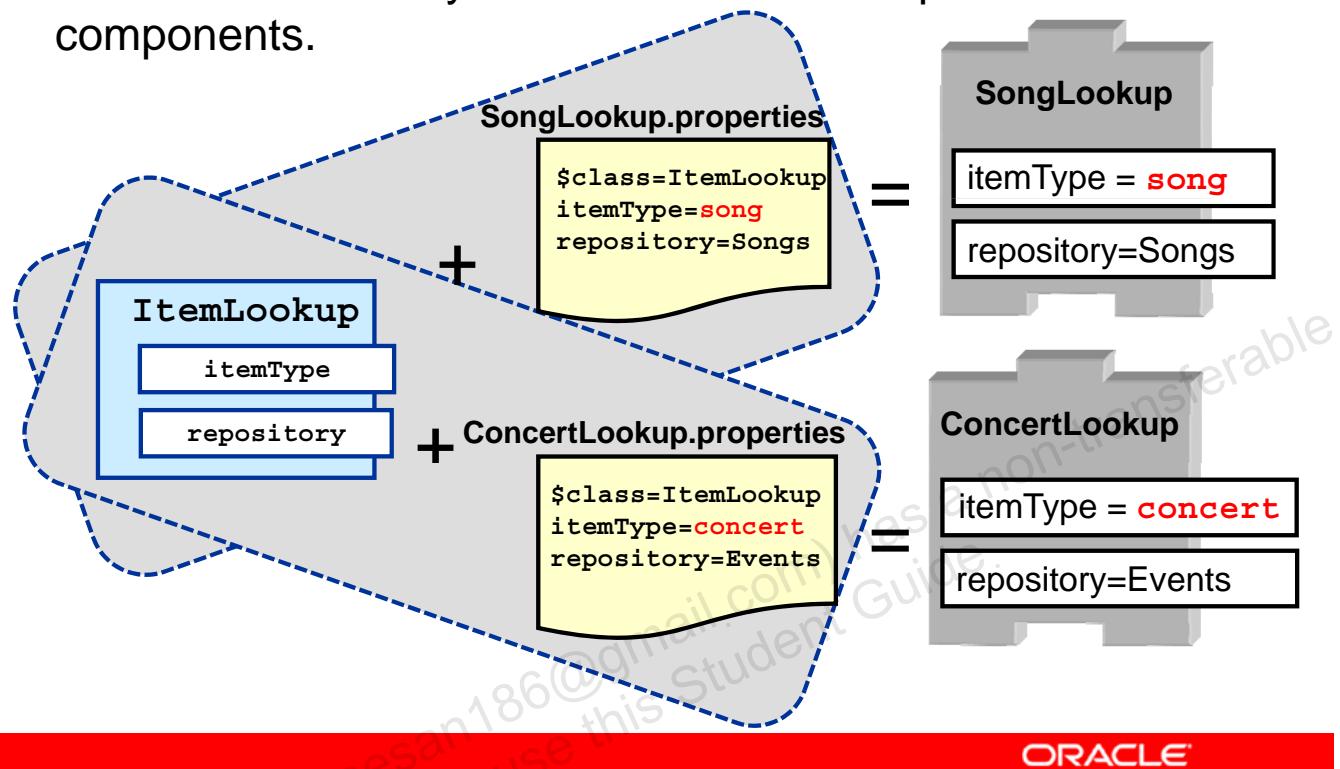
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Remember that there can be many *instances* of a component, each one a Java object that is an instance of the component's class. In each case, the configured values in the .properties file become the *initial* values for properties of the "live" component. After initialization, the live values may change, depending on usage of the particular property. For example, the Profile component (a delivered component functioning as a live representation of a user's attributes during a session) has a property called transient, which is initialized to true. This corresponds to the state of an anonymous user. When a user registers (or logs in), the live value of the transient property changes to false.

Note that for many properties, the live value never changes from its initialized value. Consider a component that provides a connection to a database. Some properties for such a component would be configured in the .properties file to point, for example, to a certain driver file and to the database server itself on the network. In cases such as these, the live properties should generally retain their initial values for the entire life of a component instance.

## Multiple Components from One Class

One Java class may be the source of multiple components.



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

ORACLE

Remember that there can be multiple *instances* of each of the components. For example, referring to the diagram in the slide, the server may at some snapshot in time have three live instances of Component 1 (`ConcertLookup`), as well as 10 live instances of Component 2 (`SongLookup`). All of these instances are objects of the same Java class, but are initialized according to one or the other of the `.properties` files shown.

## Configuring Components

You can create components and configure initial properties by:

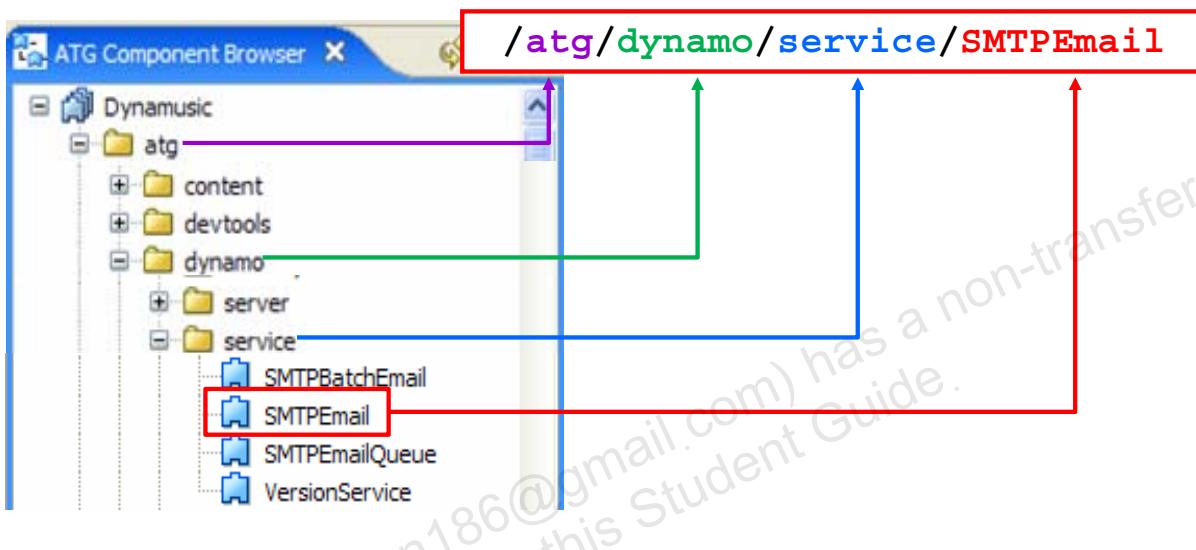
- Using the Eclipse ATG component browser plug-in
- Using the ATG Control Center component browser
- Manually editing the .properties text file



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

## Component Names

*Full component names* are unique, containing namespace names followed by the simple component name, separated by forward slashes.



ORACLE

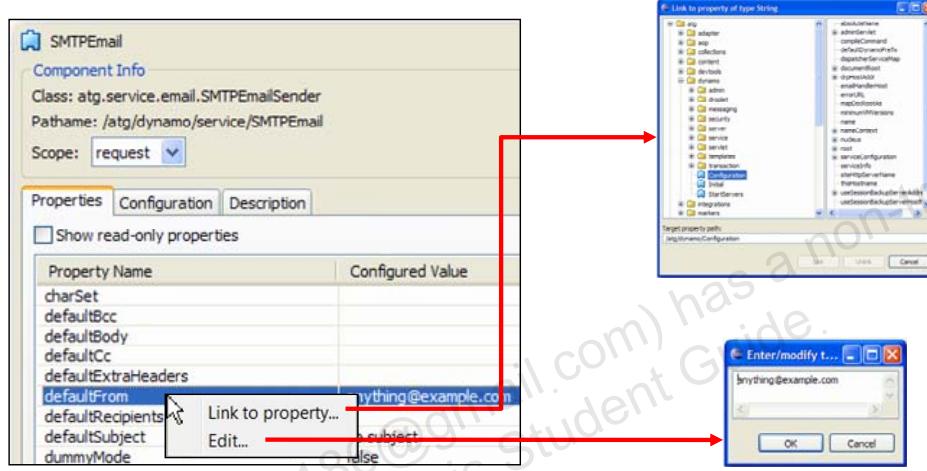
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The component shown in the slide is the result of `SMTPEmail.properties`, located in the `/atg/dynamo/service` folder structure. *Full component names* are unique, containing namespace names followed by the simple component name, separated by forward slashes. The simple component name is the same as the name of the properties file that defines it. Namespace names are created from the folders under which the properties file resides, relative to a CONFIGPATH location. The full component name of the example in the slide is `/atg/dynamo/service/SMTPEmail`.

## Instructor Demonstration: Components

### Viewing and modifying components

- Browsing components
- Setting properties (single and multi-value)
- Linking and unlinking components



ORACLE®

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

## Properties Files

Property Name	Configured Value
howMany	4
itemDescriptor	song
loggingDebug	true
loggingError	true
loggingInfo	true
loggingWarning	true
nucleus	
random	false
repository	/dynamusic/SongsR

```
# /dynamusic/FeaturedSongs
#Mon Apr 23 09:39:14 EDT 2011
$class=dynamusic.SongList
$description=A simple example
component.
$scope=global
howMany=4
itemDescriptor=song
songIds=200023,200036,200064,
205201,208204,204101,
```



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

When you save a component in the Component Browser of Eclipse or the ACC, a *properties* file is created (or updated). You can also use a text editor for the *.properties* file. You refresh the view in Eclipse or the ACC to see the changes.

Properties file syntax:

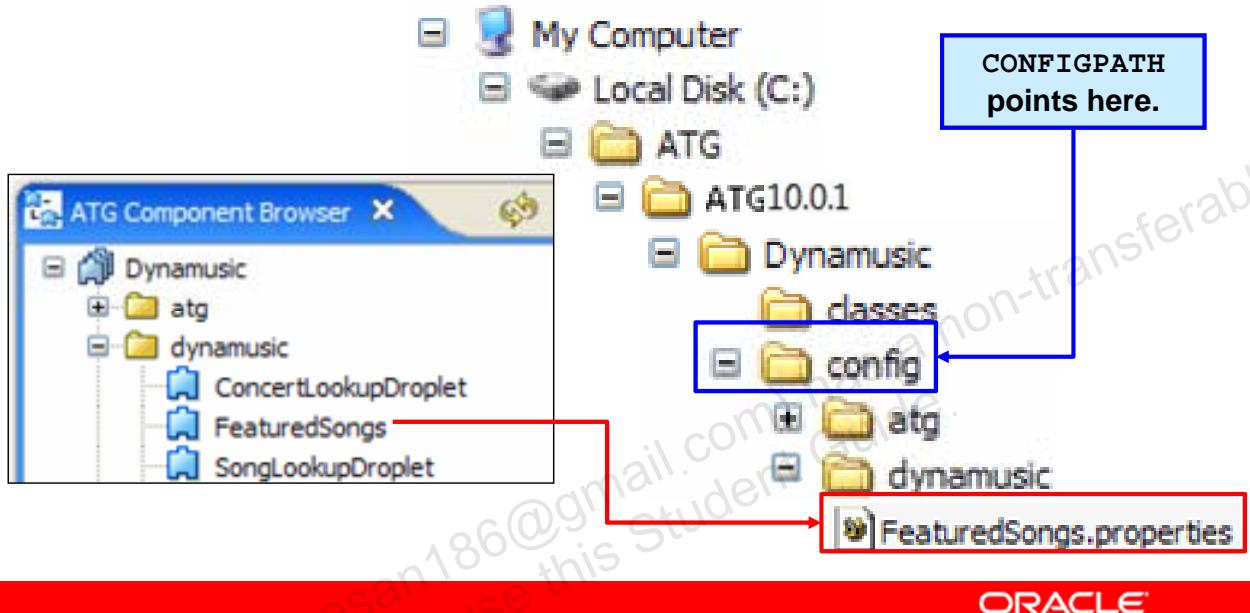
- Property-name=value
- Comment lines start with ! or #
- Special characters are entered as Unicode (\u002c) or C-style backslash-quoted (\n).
- Spaces are ignored except within or after the property value.
- A comma is the separator character for initializing list elements (array, Set, and so on).
- \ at the end of line continues the property to the next line.
- \$ indicates configuration information for the component itself, apart from values of the properties.
- += and -= can be used with list elements when using CONFIGPATH layering (discussed later in this lesson).

See notes on next slide for an example of initializing a Map.

## CONFIGPATH Environment Variable

ATG looks for .properties files in directories specified by CONFIGPATH, for example,

CONFIGPATH=<atgdir>/Dynamusic/config



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

ORACLE

A Map property can be initialized with multiple lines:

```
Property-name=\  
    key1=value1,\  
    key2=value2
```

The example in the slide show the /atg/dynamusic/FeaturedSongs component. The CONFIGPATH points to the config directory under the Dynamusic module (C:/ATG/ATG10.0.1/Dynamusic/config in the slide).

## CONFIGPATH Environment Variable

- CONFIGPATH actually points to many folders.
- Nucleus reads the entire CONFIGPATH from left to right.
  - Every matching properties file is loaded.
  - If the same property value is defined in two files, the last value overrides previous values.
    - This is called *configuration layering*.

ORACLE®

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

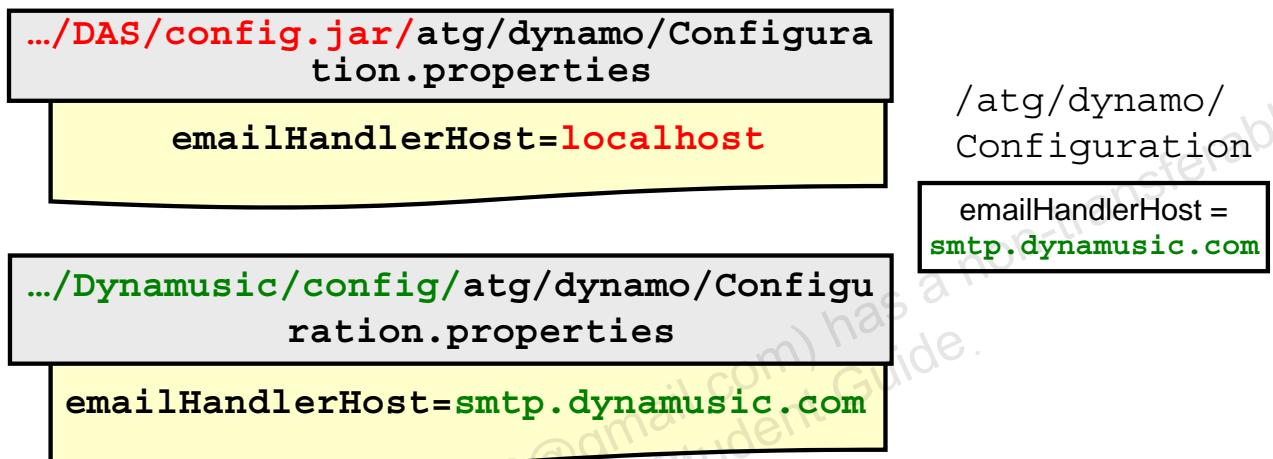
“Loaded” means retrieved and read by Nucleus.

Note that one entire file does not get replaced by another. Rather, individual properties override (or are appended to) those set earlier. Examples of this “properties file layering” are given in the following slides.

## Example: CONFIGPATH Layering

CONFIGPATH:

```
<atgdir>/DAS/config/config.jar;
<atgdir>/Dynamusic/config;
<atgdir>/home/localconfig
```



**ORACLE**

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

In the example in the slide, CONFIGPATH contains three entries:

- The DAS module configuration directory (stored in a .jar file, but shown here in its expanded form)
- The Dynamusic module configuration directory
- The localconfig directory (which is typically last—this directory should be used mostly by system administrators for deployment purposes; values in properties files in this directory override properties set in all ATG module directories)

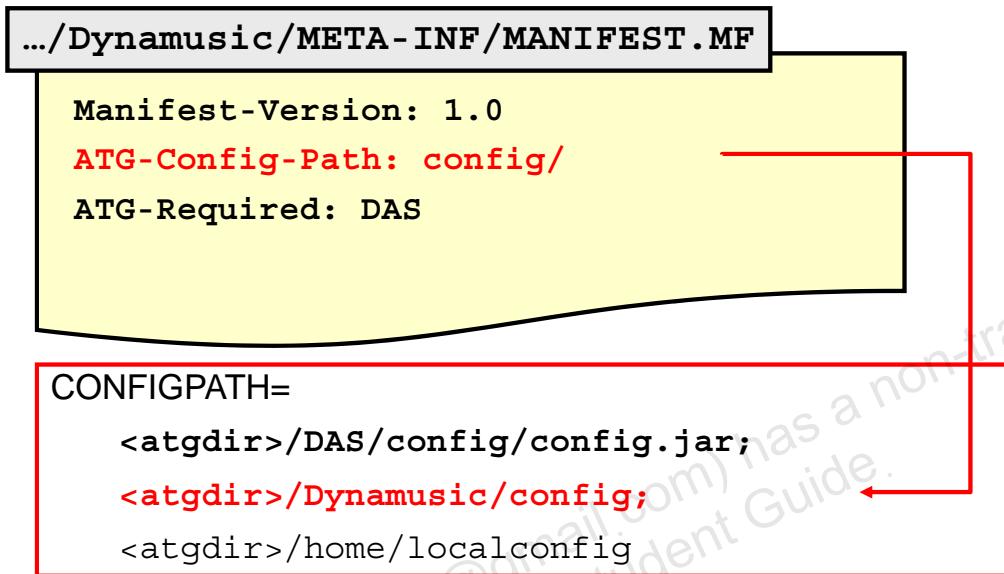
Two properties files with the name Configuration.properties contain different values for the emailHandlerHost property. Because the Dynamusic/config directory is later in the CONFIGPATH than DAS/config, the value specified in the Dynamusic directory for emailHandlerHost overrides the value for the same property set in the earlier DAS directory.

Another real-life example of using CONFIGPATH layers: You might have a JTDataSource component pointing to a particular datasource in development, but system administrators override that to point to a different datasource in production.

One of the features of configuration layering is that, for list (array, Set, and so on) properties, you can *append* values rather than just override them in layers. The syntax to do this in a properties file is `listproperty += value-to-append`. You can also remove elements using `-=`.

## Setting CONFIGPATH

Each module sets its own CONFIGPATH entry (or entries) in the module manifest file.



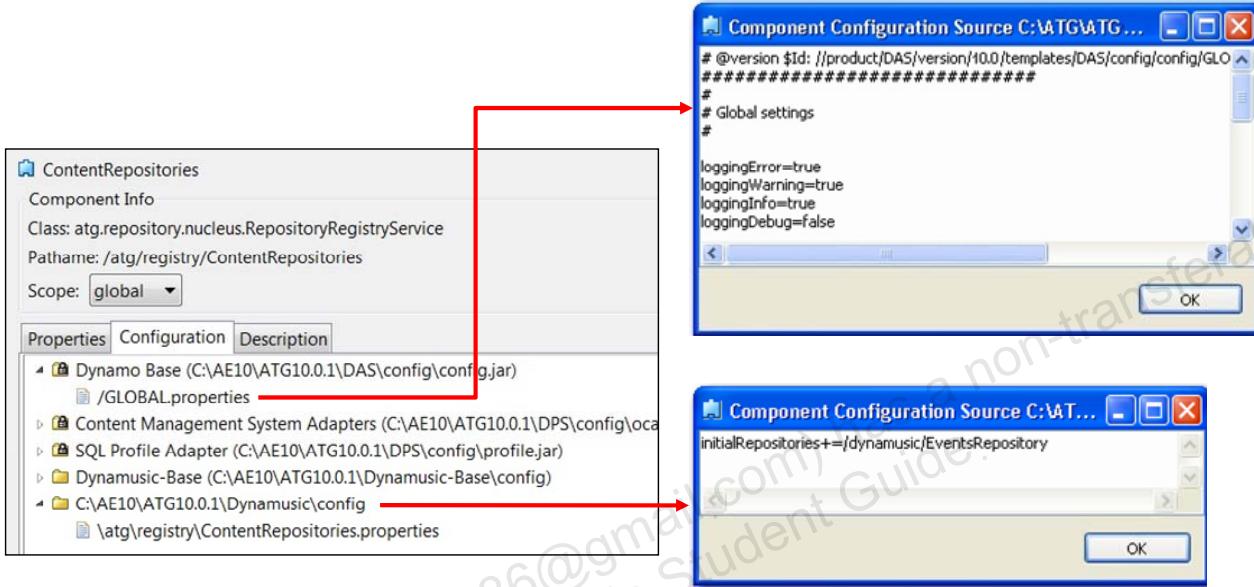
ORACLE

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

You may recall that when you created a new ATG application module using the wizard in Eclipse, the setup dialog box included a setting for the module's config directory. Although you can name this directory anything, ATG recommends you follow the convention of calling it config and placing it in the top level of the module.

## Viewing Configuration Layers

The Eclipse Component Browser enables you to view all the files in a component's configuration.



ORACLE

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

When you double-click one of the file names, a read-only viewer displays the file in Eclipse. Note that the Component Editor in the ACC behaves similarly.

## CONFIGPATH Layer Usage

Where	Who	What
.../DAS/config .../DPS/config .../DSS/config .../DCS/config Etc...	ATG	ATG products
.../module/config	<b>developers</b>	application module
.../module/liveconfig	administrators	application module
.../home/localconfig	administrators	site



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

- **Product layers:** Every ATG product module adds one or more layers to the CONFIGPATH. These config layers are in read-only JAR files. Developers and system administrators should never modify configuration layers owned by ATG. Any changes to those layers are overwritten when a new version of ATG is installed.
- **Module layers:** In general, developers are responsible for settings within your application modules.
- **Liveconfig layers:** Administrators can set properties specific to a production, rather than a development, environment.
- **Localconfig layers:** These layers are for administrators to control site-wide, cluster-wide, or server-specific settings.

Other important types of layers include:

- .../home/servers/server-instance-name/localconfig, and
- .../app-server-distribution/ATG-Data/localconfig

The first distinguishes low-level server instance configuration and the second is created and used in an application server directory when an ATG stand-alone EAR is run. Both of these are discussed in more detail in the *Configuring and Deploying an ATG Oracle ATG Web Commerce Site* course and the *ATG Installation and Configuration Guide*.

## Component Instantiation

- When Nucleus creates a new instance of a component, it
  - Looks in CONFIGPATH folders for properties files
  - Creates a live component instance, setting values from properties files
  - Starts other components if those components are being referenced by the component whose instance the Nucleus created
- Component instantiation can be tested in the ACC using the  (start) and  (stop) buttons.



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

When a new live component needs to be instantiated, Nucleus:

- Locates the properties files for the component in the directories specified in CONFIGPATH
- Finds (and loads, if necessary) the class of the component
- Calls the class constructor
- Calls the set methods of the bean, passing the initial values specified in the properties file

Locating a component is referred to as “name resolution.”

The .properties file is also referred to as the “configuration file.”

You can instruct Nucleus to start a component at server startup time. You might do this if you have a component that takes a long time to start or occasionally fails when started. To do this:

- Modify the /Initial component by either using the ACC or modifying Initial.properties
- Append the new component reference to the initialServices property

## Configured Versus Live Property Values

- Property values on new component instances are *initialized* from properties files.
- *Live* values can be changed programmatically without changing the properties file.
- For testing, view or change live values in ACC or Admin UI.

atgDynamo Component Browser  
admin/Dynamo Component Browser

**Service /dynamusic/FeaturedSongs/**

Class dynamusic.SongList

[View Service Configuration](#)

**Service Info**

SongList

**Properties**

Name	Value	Type
absoluteName	/dynamusic/FeaturedSongs	java.lang.String
adminServlet	instance of atg.nucleus.GenericService\$1AdminServlet.java: servlet	

ORACLE

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

Live values are lost when the component is destroyed.

## Component Scope

Component scope may be:

- Global (the default)
- Session
- Request
- Window
- Prototype



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

An application component can be set to one of the following scopes:

- **Global:** The component is shared among all users.
- **Session:** Separate instances of the component are provided to each user.
- **Request:** Separate instances of the component are provided to each active request.
- **Window:** Separate instances of the component are provided to each browser window.
- **Prototype:** Separate instances of the component are provided each time the component is resolved.

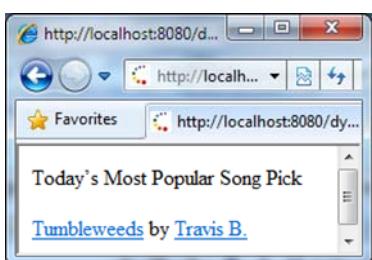
Use prototype scope to repeatedly create configured instances of a component in your application. For example, if a JSP has five separate `<dsp:valueof ...>` tags and each of them refers to the same prototype-scoped component, Nucleus will create five instances (objects) of that same component when the JSP is rendered. Also, if two components reference the same prototype-scoped component, Nucleus creates a separate instance for each of them. Prototype scope is used for some of the components that work with Endeca cartridge handlers.

You cannot set Window or Prototype scope in the ATG Eclipse plug-in.

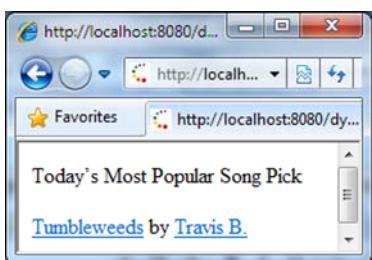
## Global Scope

- One instance of the component is shared by all users of the application on a server.
- The state of the component is the same for all users.

Browser/User #1



Browser/User #2



ORACLE

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

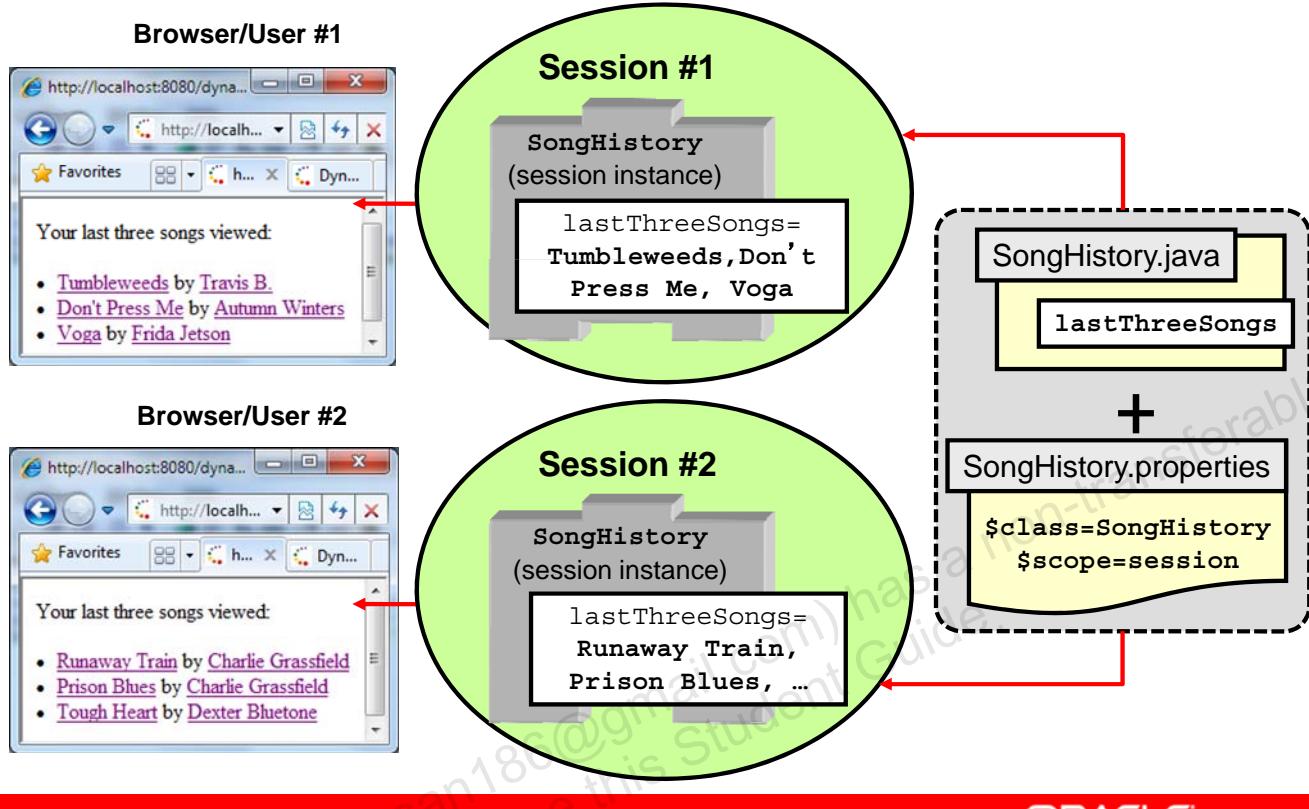
Global components are accessible from all other components.

The component is not released until the ATG server process terminates.

Users on different browsers can manipulate the same object, and possibly overwrite each other's work. To prevent this, consider synchronizing methods that set values.

For those familiar with Java EE, at first glance the global scope might appear to be the equivalent of the Java EE Application Scope, but it is not. Components with global scope are available across applications. They are not, however, available across ATG Server instances—each server instance has its own copy of a globally scoped component. Although the example shown in the slide stores visible data, the most common use of globally scoped components is not data holding, but services that are used by other components in the ATG architecture.

# Session Scope



ORACLE

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

“Session-scoped” means that every user of the application gets a separate copy of the component, and the component exists for the duration of the user’s session.

## What Is a Session?

Sessions are special server-side objects corresponding to different users or browsers. Both the application server and ATG manage session data.

When the first request for a new session comes in, the application server creates a session object corresponding to that browser. Any session-scoped components referenced by requests from the browser within that session are placed in the session object.

ATG then uses cookies or rewritten URLs to identify requests from the same browser as belonging to that session.

If the browser has not made any requests after a certain period of time, ATG invalidates the session. The default is 30 minutes.

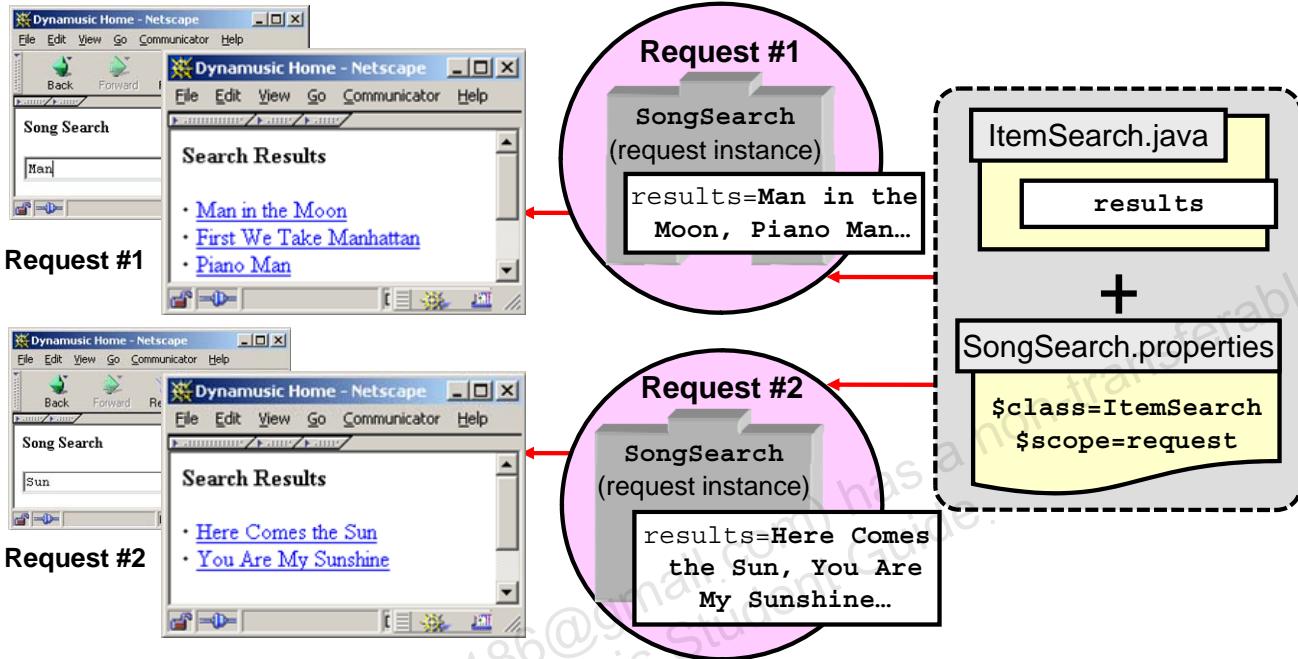
Sessions are visible in the Admin UI under:

/atg/dynamo/servlet/sessiontracking/GenericSessionManager

In the example in the slide, the SongHistory component is defined in Nucleus as /dynamusic/SongHistory. When Session #1 requests that component, a new object is created and then used throughout that session. When Session #2 requests the same component, a new object is created (such as /atg/dynamo/servlet/sessiontracking/GenericSessionManager/BCESEG8ZPU/dynamusic/SongHistory).

# Request Scope

Each request gets a different copy of the component.



ORACLE

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

Examples where request scope is useful:

- A search form
- Looking up a stock price
- Processing an HTML form (saves you from having to clear out form entry fields)

In the example in the slide, the same user (in the same browser) requests the same page ("Search Results") twice (as a result of clicking "Search" on the song search page. The first time, a query request of "man" is supplied; the second time, "sun." Each time, Nucleus creates a separate instantiation of the SongSearch component, and uses that instance on the page. (The instance goes away after the requested page is processed. The contents of the component are temporary.)

Embedded pages served in the same request share the same component instance.

Form handlers, which will be discussed in detail in the lesson titled "Using Repository Form Handlers," are often request-scoped.

## Example: Scope

- Imagine components that:
  1. Keep track of how long a user has been on a site
  2. Keep track of how many users are on the site
  3. Look up an item in a database
  4. Handle form input from the user
- What scope might you give a component doing task #1? #2? #3? #4?

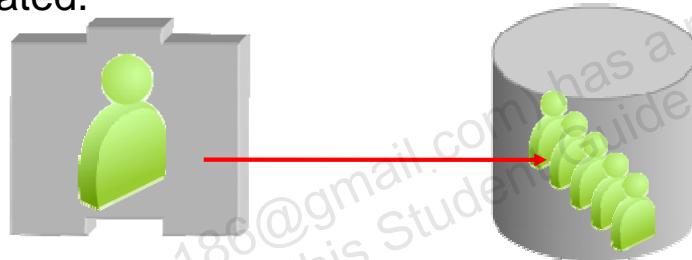
 ORACLE

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

You cannot create a reference to a component that has a scope more restrictive than the component that is referencing it. For example, if you have a globally scoped component, it cannot contain a property that links to a session-scoped component.

## A Prominent Session-Spaced Component

- The **Profile** component is central to all personalization on the ATG platform:
  - The full name is /atg/userprofiling/Profile.
  - It is intimately connected to the **Profile Repository**, which stores information for registered users.
- The Profile component is a *live* representation of a user's information.
- At the start of a session, the Profile component is unpopulated.



ORACLE

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

The Profile component, and the associated `ProfileAdaptorRepository`, are covered in much more detail in later lessons. For now, it is important to understand only how to find and use profile properties, because they will be used in several practices.

## **Instructor Demonstration: SessionManager**

Explore session-scoped components:

- The container for sessions is `/atg/dynamo/servlet/sessiontracking/GenericSessionManager`.
- Start two sessions with two different browsers (for example, open a new browser and go to `http://localhost:7103/dynamusic`).
- Notice a session component is created for each, with its own copy of the Profile component.



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

The Profile component has special functionality. It is instructive to note the `transient` property and `repositoryId` property. When a new session is first created, `transient` starts out as `true` and the `repositoryId` is an arbitrary number. When a user in a session logs in, both properties change: `transient` changes to `false`, and the `repositoryId` matches that user's repository ID. Repositories are discussed in the lesson titled "Using ATG Repositories."

If an ATG application is running, you can view sessions by navigating in the Admin UI. When running with WebLogic, use a browser to access `http://localhost:7103/dyn/admin` and click the link for the Component Browser. Then navigate to the `/atg/dynamo/servlet/sessiontracking/GenericSessionManager` component and click the "View" button. This shows the sessions running on your server instance.

# Logging

- Most components can log information about their status.
- It is mainly used for troubleshooting.



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

Any component whose base class implements the ApplicationLogging interface can use the ATG logging infrastructure. Logging can be done on a per-component, per-module, or per-application basis.

#### **Per-component:**

- For a developer to debug component implementation
- For an administrator to track what methods are being called and when

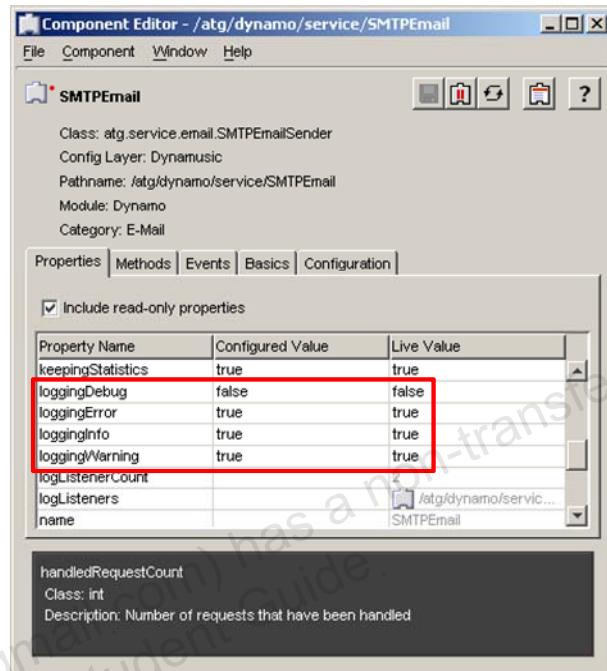
#### **Per-module or application-wide:**

- For an administrator to track site activity

Implementing logging in Java code is covered in more detail in the *Extending the ATG Application* course.

## Types of Logging Messages

- **Error:** Indicates an immediate problem
- **Warning:** May indicate a future problem
- **Info:** Events that occur as normal operation of the component
- **Debug:** Events specific to internal workings of the component



ORACLE®

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

Error, Warning, and Info-level messages are logged by default.

The error log captures all the stack trace errors generated from within ATG, such as `NullPointerException`.

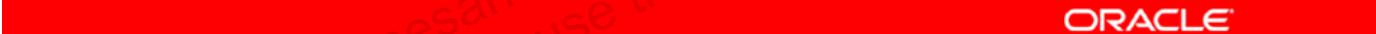
Debug-level messages are not logged by default. Debug-level messages should be needed only for debugging situations.

An example of an Info-level message includes server messages indicating handled requests.

Having messages display on the console is helpful during debugging time but should generally be removed at production time.

## Log Files

- By default ATG messages are captured in log files in <atgdir>/home/logs.
  - info.log
  - warning.log
  - error.log
  - debug.log
- You can also view ATG messages merged with messages from the app server.

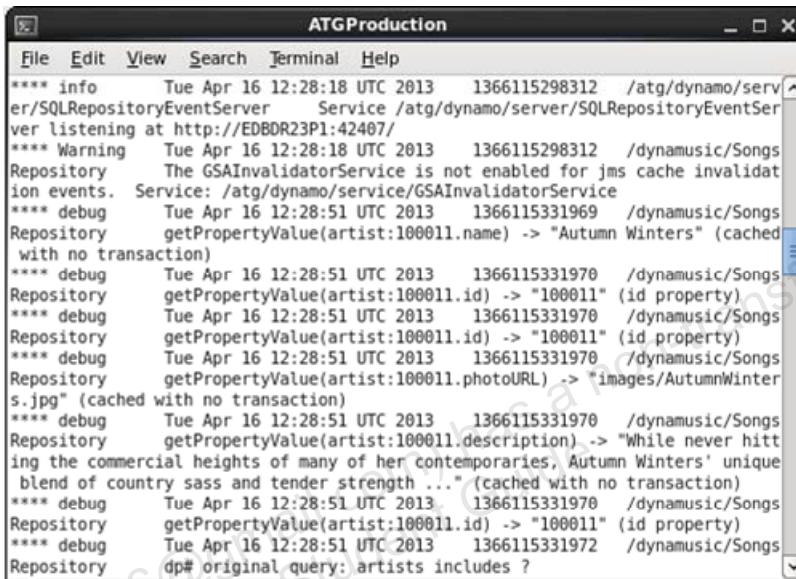
ORACLE

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

Where logging messages are sent can be modified by setting the `logListeners` property in a `GLOBAL.properties` file.

## Console Logging

By default, info, warning, and error messages are also directed to the console.



```
File Edit View Search Terminal Help
**** info   Tue Apr 16 12:28:18 UTC 2013  1366115298312  /atg/dynamo/server/SQLRepositoryEventServer  Service /atg/dynamo/server/SQLRepositoryEventServer listening at http://EDBDR23P1:42407/
**** Warning  Tue Apr 16 12:28:18 UTC 2013  1366115298312  /dynamusic/SongsRepository  The GSAINvalidatorService is not enabled for jms cache invalidation events. Service: /atg/dynamo/service/GSAINvalidatorService
**** debug   Tue Apr 16 12:28:51 UTC 2013  1366115331969  /dynamusic/SongsRepository  getPropertyValue(artist:100011.name) -> "Autumn Winters" (cached with no transaction)
**** debug   Tue Apr 16 12:28:51 UTC 2013  1366115331970  /dynamusic/SongsRepository  getPropertyValue(artist:100011.id) -> "100011" (id property)
**** debug   Tue Apr 16 12:28:51 UTC 2013  1366115331970  /dynamusic/SongsRepository  getPropertyValue(artist:100011.id) -> "100011" (id property)
**** debug   Tue Apr 16 12:28:51 UTC 2013  1366115331970  /dynamusic/SongsRepository  getPropertyValue(artist:100011.photoURL) -> "images/AutumnWinters.jpg" (cached with no transaction)
**** debug   Tue Apr 16 12:28:51 UTC 2013  1366115331970  /dynamusic/SongsRepository  getPropertyValue(artist:100011.description) -> "While never hitting the commercial heights of many of her contemporaries, Autumn Winters' unique blend of country sass and tender strength ..." (cached with no transaction)
**** debug   Tue Apr 16 12:28:51 UTC 2013  1366115331970  /dynamusic/SongsRepository  getPropertyValue(artist:100011.id) -> "100011" (id property)
**** debug   Tue Apr 16 12:28:51 UTC 2013  1366115331972  /dynamusic/SongsRepository  dp# original query: artists includes ?
```

ORACLE

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

The console log is an invaluable tool for doing the course practices. If you experience any unexpected behavior, you should check the console for possible errors. If you find the log scrolls by too quickly, you can also refer to the log files mentioned previously.

Which messages go to the screen and which go to log files can differ. For example, this is controlled by JBoss, which uses log4j to manage logging. By default, only info, warning, and error messages are logged to the console screen. To see debug messages, you need to look in the debug.log file. WebLogic logs debug messages to the console screen by default.

Many ATG developers use a handy open source utility called ATGLogColorizer to highlight different types of messages. Although this course does not specifically cover or depend on ATGLogColorizer, you may use it if you want. You can download the utility from <http://atglogcolorizer.sourceforge.net/>. To use it, pipe the startup script output through the utility, as in

```
startDynamoOnJBOSS -c atg_foundations -f -run-in-place -m Dynamusic | /path/to/ATGLogColorizer.exe
```

## GLOBAL.properties

GLOBAL.properties is a special properties file, not associated with any specific component, often used to set logging levels for sets of components.

.../config/GLOBAL.properties

```
loggingError=true  
loggingWarning=true  
loggingInfo=true  
loggingDebug=false
```



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

Any directory or subdirectory under the CONFIGPATH can have a GLOBAL.properties file, and it affects all components in that subdirectory and all directories under that subdirectory. The properties set in GLOBAL.properties are set for all Nucleus components that have properties whose name matches a setting in the GLOBAL.properties files, unless the component's own .properties file sets that property explicitly.

The most common use of the GLOBAL.properties file is to set logging levels for ATG subsystems. You might want to set loggingDebug to true for all the components in your application by setting that property in <yourmodule>/config/GLOBAL.properties, or for some subset of components, such as

<yourmodule>/config/atg/dynamo/dms/patchbay/GLOBAL.properties, if you are debugging some functionality in Patchbay.

At production time, it is recommended to set the properties in your top level GLOBAL.properties file like this:

```
loggingError=true  
loggingWarning=true  
loggingInfo=false  
loggingDebug=false
```

The Info and, especially, the Debug log levels generate very large numbers of messages and could impair the performance of a live site.

# Quiz

\_\_\_\_\_ scope provides separate instances of a component to each user.

- a. Global
- b. Session
- c. Request
- d. Window
- e. Prototype



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

**Answer: b**

## Quiz

\_\_\_\_\_ scope provides separate instances of a component each time the component is resolved.

- a. Global
- b. Session
- c. Request
- d. Window
- e. Prototype



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

**Answer: e**

## Quiz

\_\_\_\_\_ scope provides separate instances of a component to each active request.

- a. Global
- b. Session
- c. Request
- d. Window
- e. Prototype



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

**Answer: c**

## Quiz

When Nucleus loads configuration files in the CONFIGPATH (choose all that apply) :

- a. Configuration file names must end in .properties.
- b. Only one properties file is loaded if there are matching properties files.
- c. You can only create properties files in Eclipse.
- d. If the same property value is defined in two files, the last value overrides previous values.



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

**Answer:** a, d

# Quiz

Which of the following are valid logging levels?

- a. Info
- b. Request
- c. Warning
- d. Error
- e. Fix
- f. Debug



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

**Answer:** a, c, d, f

## Summary

In this lesson, you should have learned how to:

- Create and configure Nucleus components
- Work with CONFIGPATH layering to modify component configuration in modules
- Set the appropriate component scope
- Use component logging properties

## For More Information

For more information, refer to the chapter titled “Nucleus: Organizing JavaBean Components” in the *Programming Guide*.



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

## Practice 2 Overview: Using Components

These practices cover the following topics:

- Modifying an existing component (`SMTPEmail`) using the Component Browser in Eclipse
- Creating and configuring a new `FeaturedSongs` component based on a supplied class



ORACLE

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

The `FeaturedSongs` component is used in the practices for the lesson titled “Dynamic Pages.”

# Dynamic Pages

ORACLE®

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

# Objectives

After completing this lesson, you should be able to:

- Create JavaServer Pages that use the DSP tag library to access ATG functionality
- Embed page fragments into pages and pass parameters between pages
- Use Dynamo Servlet Beans to generate page output

## Dynamusic.com Pages

Dynamusic needs dynamic pages to:

- Display the song, artist, and album information from a repository
- Display personalized content based on user registration information
- Provide forms for users to enter information



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

# Road Map

- JavaServer page basics
- JSPs and ATG
- DSP component access
- DSP page parameters and fragments
- Dynamo Servlet Beans
- Nested DSP tag restrictions



ORACLE

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

## JSP: Overview

- What is JSP?
  - A language for creating dynamic web pages
  - An extension of Java Servlet technology
- JSP:
  - Uses XML-like tags to insert dynamic content within HTML
  - Is deployed within a Java EE web application
  - Supports direct inclusion of Java code on a page or interaction with separate Java classes



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

JSP is a markup language for dynamic web pages. JSPs are compiled into Java code (servlets), which is run whenever a dynamic page is requested. The servlets generate HTML that is returned to the client via a response object.

This course will present the basics on JSP: syntax and a few necessary JSP tags. It is not intended to provide an in-depth understanding of JSP, only enough JSP to allow page developers to fully utilize ATG features. Developers working with JSP should strengthen their understanding of JSP through any number of excellent resources available.

## JSP Review: Syntax

- JSP directives: `<%@ ... %>`
- JSP comments: `<%-- ... -->`
- JSP tags are marked by `jsp:tagname` and use XML-like syntax:

```
<jsp:tagname attributes... />
or
<jsp:tagname attributes... >
    tag body
</jsp:tagname>
```



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

Directives are executed at compile time (when the page is converted to a Java Servlet). For example, you can import Java packages and classes using the page directive:

```
<%@ page import="java.util.Vector" %>
```

Tags are Java classes that are called by the page at run time. JSP comes with a set of tags, accessed using `jsp:..`. Custom tags can also be created, but will not be covered as part of this course. ATG provides a tag library called Dynamo Server Page (DSP) tag library, which will be discussed later in this lesson.

JSP comments are not visible in the HTML generated by the servlet. Some JSP developers therefore use HTML comments which are in the generated HTML (and can contain JSP code) to help with debugging. For instance, you can embed a date within an HTML comment using

```
<!-- <%= new Date() %> -->.
```

## JSP Review: Working with Objects

- Variable declarations: Variables can be used by expressions or scriptlets.  
`<%! SongList slist=new SongList()%>`
- Expressions: They return a single value.  
`Current song is <%= slist.getNext()%>`
- Custom tag library tags can also provide access to objects.  
`<dsp:valueof param="element.name"/>`

ORACLE

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

Variables are declared using standard Java syntax. Multiple declarations may be in the same tag:

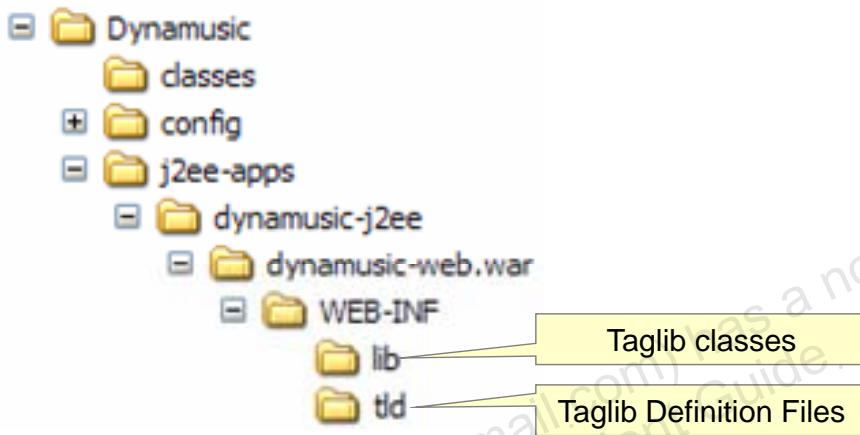
- `<%! String name = "Jane";  
Date currentDate = new Date();  
int count=0;  
float value1, value2, value3; %>`

Expressions can invoke methods on JavaBean components, access scripting variables, and access implicit objects (such as the page request object)

The DSP tag library, supplied by ATG and covered later in this lesson, allows JSPs to access Nucleus components.

## JSP Review: Using Tag Libraries

- Tag libraries allow pages to invoke separate Java classes.
- Declaration: `<%@ taglib uri="..." prefix="xxx" %>`
- Usage: `<xxx:tagname ... > ... </xxx:tagname>`



ORACLE

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

Tag library classes are included in the Java EE web application in JAR files, and defined in the Tag Library Definition (TLD) files. The application server scans the WEB-INF directory for TLDs.

Taglib URI: Declared in `web.xml`. It associates a URI with a tag library descriptor file (`.tld`).

Prefix: This is an arbitrary string used to identify tags within the tag library that are used on this page. Although this string could be anything you choose, for convenience and clarity we will be using the `dsp` prefix through the course, and you are recommended to follow this convention. For example, note that the page tag is referenced as `<dsp:page>`; had you used a different prefix, such as `foo`, the tag would have been referenced as `<foo:page>`. As you can see, this can get very confusing if you change the tag library prefix from page to page within your application.

## JSP Review: JSTL

- The JSP Standard Tag Library (JSTL) provides common JSP functionality such as iteration, comparisons, internationalization, and XML document manipulation.
- Standard tags come in two syntax flavors:
  - Run (or request) time (RT): Default
  - Expression language (EL): Allows use of simplified access to Java object properties

```
<c:out value="${request.cookies}[0]>
```



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

The ATG Eclipse plug-in J2EE wizard automatically includes and defines all the JSTL tag libraries, as well as the ATG DSP tag library (discussed shortly).

JSTL includes several sets of tags, including:

- **core (c)**: Common programmatic functions such as iteration and comparisons
- **format (fmt)**: Output formatting
- **XML (x)**: XML manipulation

Frequently used core tags include:

- **set**: Sets an attribute (variable) or JavaBean property
- **out**: Displays an attribute or property
- **foreach**: Loops over a set of values
- **if**: Comparison
- **choose/when**: Switches on a value

JSTL Expression Language supports “dot” notation for JavaBean properties:

- ```
<c:out value="${SMTPEmail.emailHandlerHostName}" />
```

JSTL also supports numerous operators (==, >=, +, &&, and so on).

You can learn about JSTL from numerous places.

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

# Road Map

- JavaServer page basics
- JSPs and ATG
- DSP component access
- DSP page parameters and fragments
- Dynamo Servlet Beans
- Nested DSP tag restrictions



ORACLE

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

## DSP Tag Library

- The DSP tag library provides access to ATG functionality from JSPs.
- Declare at the beginning of the JSP:  
`<%@ taglib uri="/dspTaglib" prefix="dsp" %>`
- The contents of the page must be within the page tag:  
`<dsp:page>  
...page content...  
</dsp:page>`



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

When using Eclipse to create your ATG web application, the DSP and DSPEL tag libraries are automatically included in the web application for you. The two TLD files for these two libraries define the URLs respectively as:

- `http://www.atg.com/taglibs/daf/dspjspTaglib1_0`
- `http://www.atg.com/taglibs/daf/dspjspELTaglib1_0`

For convenience, the generated `web.xml` file creates a shorter alias URI for the DSP tag library, `/dspTaglib`. For this course, the alias is used to make reading code samples easier.

If you want to use the Expression Language version of the DSP tags, in addition to or instead of the Runtime version shown in the slide, declare the DSPEL tag library as well:

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspELTaglib1_0"  
prefix="dspeL" %>
```

Both libraries contain the same tags; the only difference is which syntax is supported for expressions.

The prefix specified in the taglib declaration is arbitrary, and you could use any string, but ATG recommends you follow the convention of using `dsp` for the RT tags and `dspe1` for the EL tags.

The `page` tag: All the DSP tags that will be discussed throughout the rest of the course must be children of the `page` tag. Although not all examples show the `page` tag (due to limited screen space), it should always be assumed that any `dsp` tag appears within a `page` tag.

## Creating and Editing JSPs

JSPs may be created and edited in:

- Eclipse: Provides highlighting and syntax completion/checking
- ACC: Provides wizards for DSP tags, droplets, and form handlers; includes personalized preview functionality
- Your favorite text editor



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

## Road Map

- JavaServer page basics
- JSPs and ATG
- **DSP component access**
- DSP page parameters and fragments
- Dynamo Servlet Beans
- Nested DSP tag restrictions



ORACLE

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

## Pages and Components

- You can use component values in dynamic web pages.
- The DSP tag library supplies tags to:
  - Display component properties
  - Set component properties



ORACLE

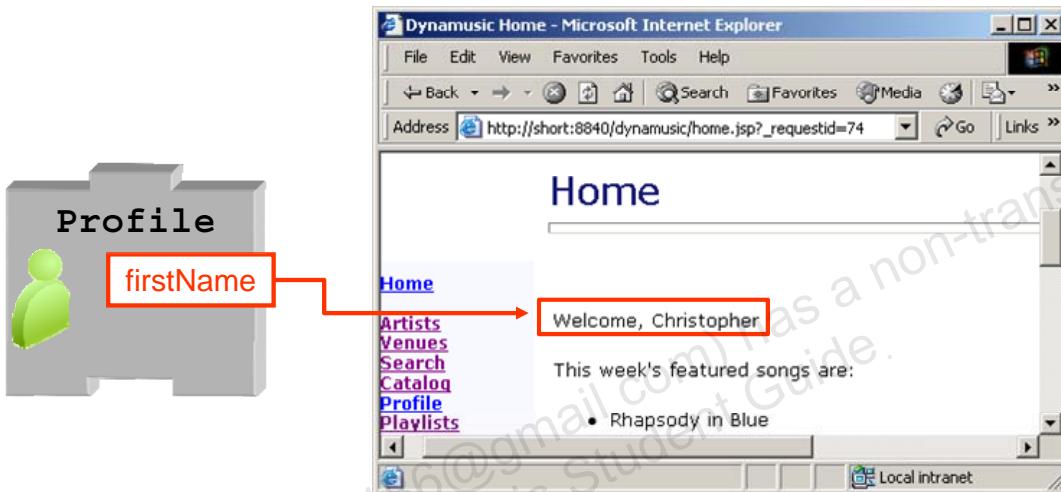
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Displaying Component Properties

Use the `valueof` tag to display property values.

Welcome, <`dsp:valueof`

```
bean="/atg/userprofiling/Profile.firstName"  
>Guest</dsp:valueof>
```



ORACLE

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

In the example in the slide, *Guest* is the default value that is displayed if the property is null. This is optional.

Remember the path is always relative to the directories specified in the CONFIGPATH. Therefore, you either:

- Import and refer to the component by name (see the next slide)
- Or
- Refer to the component with its path relative to CONFIGPATH.

If you do not need to specify a default value for your display property, you can use the following shorter syntax:

```
<dsp:valueof bean="/atg/userprofiling/Profile.firstName"/>
```

## Importing Components

- It is helpful to import components:  
`<dsp:importbean  
bean="/atg/userprofiling/Profile"/>`
- Importing a component lets you access its properties by a simple name rather than the full component path:  
`<dsp:valueof bean="Profile.firstName" />`



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

By importing the bean, the previous example could have used `<dsp:valueof  
bean="Profile.firstName" />` instead of `<dsp:valueof  
bean="/atg/userprofiling/Profile.firstName" />`.

The `<dsp:importbean>` tag has an optional `var` attribute that can sometimes prove useful, especially when using JSTL. The `var` attribute names an EL variable that makes the servlet bean instance accessible to other tags on the page. When `var` is omitted, the shorthand name for the component is used.

## Tag Converters

- Tag converters allow you to format the output of a `<dsp:valueof>` tag.
- Examples:

```
<dsp:valueof bean="Profile.dateOfBirth"  
date="MMM dd, yyyy" />
```

Birth date: May 25, 1963

```
<dsp:valueof bean="Profile.dateOfBirth"  
date="M dd yy" />
```

Birth date: 5 25 63



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

ATG provides several converters, used for formatting output in a `dsp:valueof` tag, input in a form (discussed in the lesson titled “Using Repository Form Handlers”), or both (such as `date`).

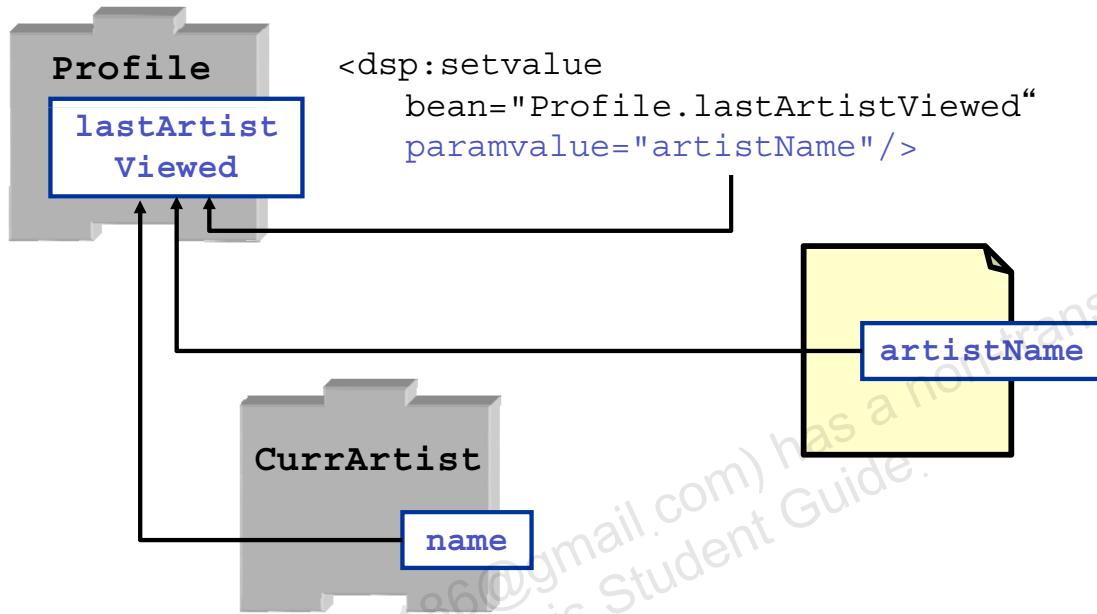
The format specifiers (such as “`MMM dd, yyyy`” in the first example in the slide) are detailed in the documentation.

Other output converters include `number`, `currency`, and `creditcard`. The `valueishtml` converter allows you to store HTML code in a component, and have the display correctly parse the HTML. (By default, HTML special characters are converted to unicode so that they display literally.) You can also create custom tag converters.

For more information about converters, see the chapter titled “Creating Java Server Pages” in the *Page Developer’s Guide*.

## Setting a Component Property

You can set a component property within a page.



ORACLE

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

The `<dsp:setvalue>` tag has three different forms, depending on the source of the value being set:

- **Fixed value:** `<dsp:setvalue bean="Profile.lastArtistViewed" value="Miles Davis"/>`
- **Component property value:** `<dsp:setvalue bean="Profile.lastArtistView" beanvalue="/dynmusic/CurrentArtist.name"/>`
- **Parameter value:** `<dsp:setvalue bean="Profile.lastArtistViewed" paramvalue="artistName"/>` (Parameters are covered in subsequent slides.)

Several other tags that manipulate values follow a similar approach, as you will see.

## Quiz

If a user, Brock, navigates to your site and he does not log in, what will the result of the following code be?

```
<dsp:importbean bean="/atg/userprofiling/Profile"/>
. . .
Welcome, <dsp:valueof
bean="Profile.firstName">Guest</dsp:valueof>
```

- a. Welcome, Anonymous
- b. Welcome, Guest
- c. Welcome, Brock



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

**Answer: b**

## Practice 3 Overview: Dynamic Pages: Part 1

This practice covers the following topics:

- Creating a home page for the Dynamusic application (Practice 3-1)
- Adding a personalized greeting to the home page by displaying the `firstName` property of the `Profile` component (Practice 3-2)

The screenshot shows a web page titled "Home". On the left, there is a vertical navigation menu with links: Home, Artists, Venues, Search, Profile, and Log Out. The "Home" link is currently selected and highlighted in blue. The main content area displays a welcome message: "Welcome, Arthur!" followed by "This week's featured songs are:" and a bulleted list: "• Famous Blue Raincoat", "• Strawberry Fields Forever", and "• ...".



ORACLE

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

There are four practices for this lesson. The first two practices, shown in the slide, relate to the section just covered. The remaining practices are described at the end of the lesson.

# Road Map

- JavaServer page basics
- JSPs and ATG
- DSP component access
- DSP page parameters and fragments
- Dynamo Servlet Beans
- Nested DSP tag restrictions



ORACLE

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

## Including Page Fragments

Embedding one JSP into another allows you to reuse portions of pages as building blocks.

```
<dsp:include  
    page="relativeURL" />  
<dsp:include  
    page="<% expression %>" />
```



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

Both the DSP and JSP tag libraries have `include` tags, that work very similarly. There are two main advantages of the DSP version of `include`:

- Both the `include` tags allow you to pass parameters, but `jsp:include` parameters must be of type `String`, whereas `dsp:include` parameters can be of any type.
- In addition to the `page` attribute, `dsp:include` also has a `src` attribute that allows you to embed pages that are not part of the same Java EE web application.

The `dsp:include` tag has two additional attributes that are not covered in this course: `flush` and `staticcharset`. Information about these and the `src` attribute can be found in the DSP tag library appendix of the *Page Developer's Guide*.

Relative page URLs are resolved relative to the including page. Absolute page URLs are resolved relative to the web application (discussed in greater detail later in the course).

**Note:** You can “hard code” a relative URL or you can dynamically calculate the desired URL by writing a JSP expression.

## Additional information about Embedding Page Fragments

### Static Includes

As an alternative to the `include` tag (which does *dynamic* page inclusion), JSP also provides a directive that *statically* includes pages: `<%@ include file="relativeURL" %>`

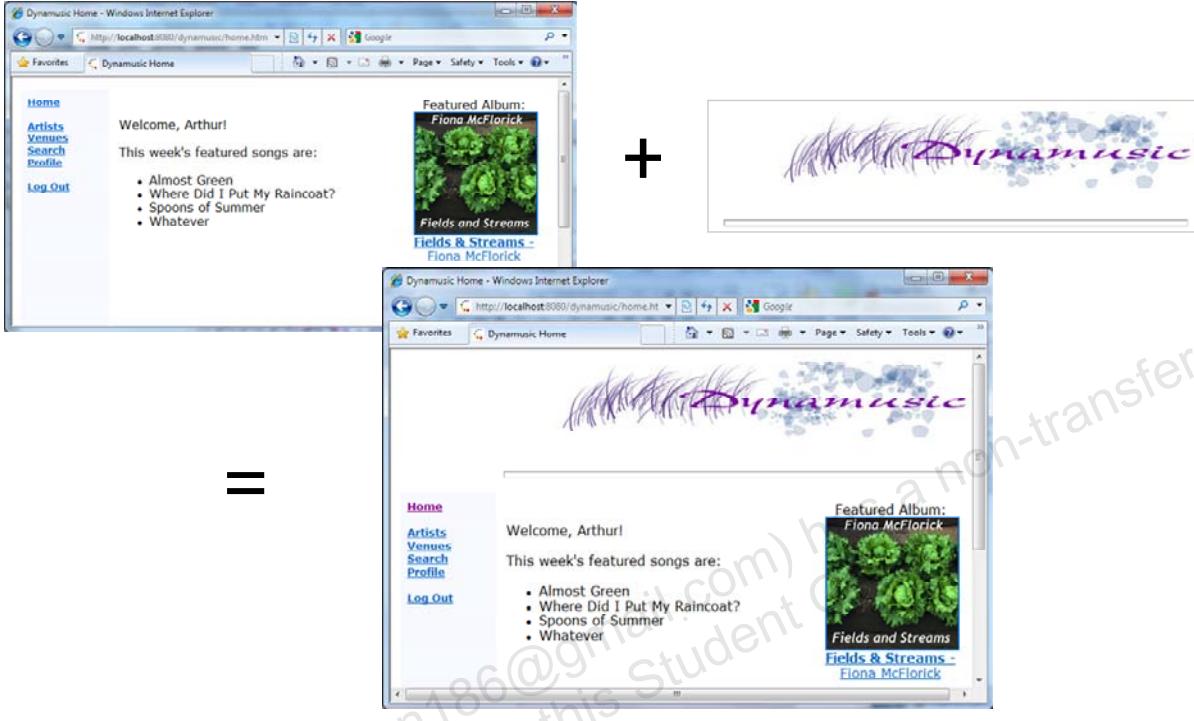
This merges the text of the included file into the text of the including file. This works similarly to the C/C++ `#include` macro. Static includes are processed at compile time. Because statically included pages are actually merged into the including page, the embedded page has access to all the parameters and scripting variables defined in the embedding page. Static includes are more efficient than dynamic includes, but do not allow passing parameters (covered shortly).

### Dynamic Includes

When one page fragment is dynamically included in another using the `include` tag, the two JSP files are *not* merged. Instead, both pages are compiled into separate servlets, and the `service()` method of the embedding page servlet calls the `service()` method of the embedded page. Dynamic includes are processed at run time. Because the including page and included page are actually separate servlets, they do not have access to each other's parameters and scripting variables.

## Example: Including Page Fragments

Useful for reusing common page snippets



ORACLE®

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

Dynamusic has a “branded” header that they would like to add to the top of each page. It includes their logo as well as some table formatting. Rather than enter the 10 or so lines of code on each page, they would rather have one single location for their header, and then include that on each page that needs it.

## Example: Including Page Fragments

### home.jsp

```
<%@ taglib uri="/dspTaglib" prefix="dsp" %>
<dsp:page>
<html>
  <body>
    <dsp:include page="common/header.jsp"/>
    ... Welcome to Dynamusic. ...
  </body>
</html>
</dsp:page>
```

### header.jsp

```

<hr size=8>
```



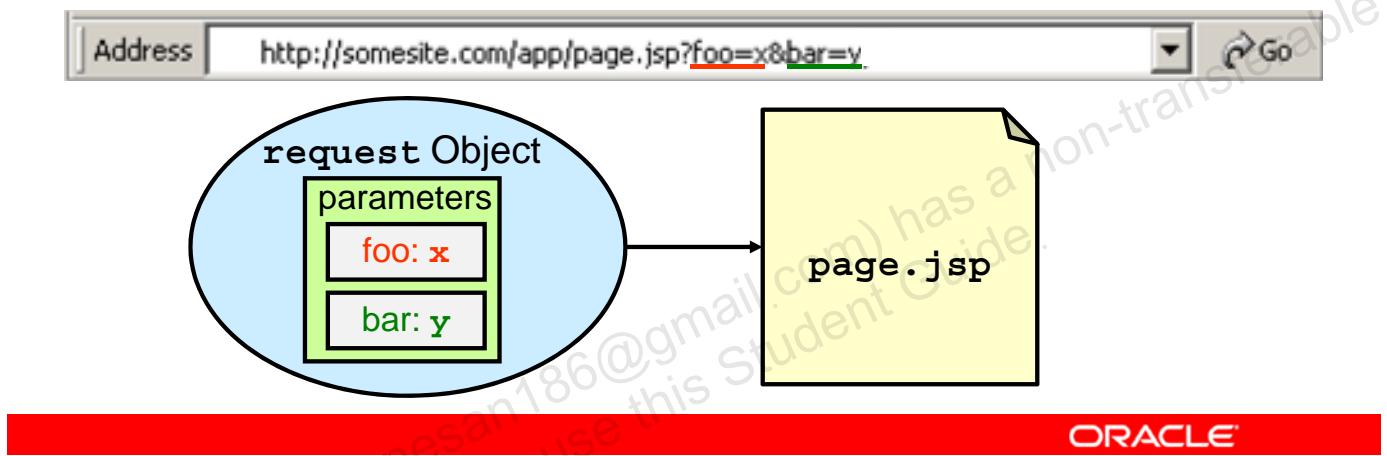
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example shown in the slide is the simplest situation: Including a page that is completely static. In the following slides, you modify this example so that the included page is dynamic and able to deal with page parameters passed from the calling page.

## Page Parameters

Parameters are variables associated with a page request that can be used in JavaServer Pages. They can be:

- Used within a page
- Passed to an embedded page
- Passed to another page via a link



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

Recall that JavaServer Pages are compiled into servlets, and that servlets have a `service()` method, and that one of the parameters to that method is `request`. The `request` object can contain parameters, either specified in the URL (as shown in the example in the slide) or passed by the previous page as part of the header (as is typically done with forms). This is the mechanism by which parameters (as well as other data associated with the `request`) are passed to a page. This is not specific to ATG, but is part of the general Hypertext Transfer Protocol (HTTP).

## Accessing Parameters

- Parameters passed to a dynamic page or page fragment are accessible anywhere within the page.
- Parameter values are accessed with the `< dsp:valueof>` tag:

```
<dsp:valueof param="param_name">  
  [default value]  
</dsp:valueof>
```



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

If you do not need to specify a default value, a shorter form can be used:

```
<dsp:valueof param="param_name"/>
```

Note that DSP parameters can be of any Object type, unlike JSP parameters, which are limited to Strings.

Note also the `<dsp:valueof>` tag can be nested inside of other non-DSP tags, as in:

```
<body bgcolor=<dsp:valueof bean="a.b"/>>
```

Note that you should *not* nest `<dsp:valueof .../>` inside a `param` tag, as in:

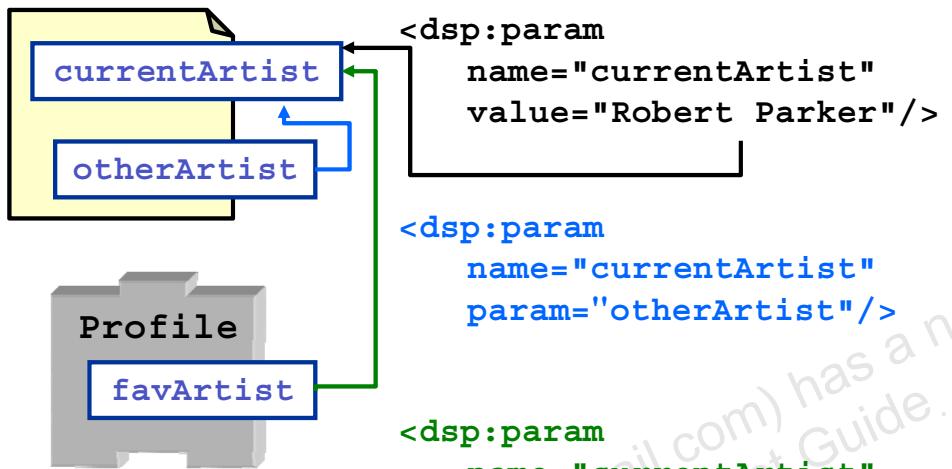
```
<dsp:param name="x" value=<dsp:valueof bean="a.b"/>" />
```

Instead, use the following type of syntax:

```
<dsp:param name="x" bean="a.b" />
```

## Setting Parameters

You can set a page parameter using a component property, another parameter, a JSP expression, or a literal.



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

Similar to the `<dsp:setvalue>` tag covered earlier, the `<dsp:param>` tag has three different forms, depending on the source of the value being set.

## Passing Parameters

- Parameter values are passed to an embedded page by placing a `<dsp:param>` tag inside the `<dsp:include>` tag.

```
<dsp:include page="pagename.jsp">
  <dsp:param name="showArtist"
    bean="Profile.favArtist"/>
</dsp:include>
```
- Parameters passed to embedded pages may be of any type (primitives or Java objects).



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

Parameter values are passed to an embedded JSP by placing a `<dsp:param>` tag within a `<dsp:include>` tag.

As with the `include` tag, both the JSP and DSP tag libraries provide a `param` tag. The difference is that DSP parameters can be of any type (Java object or primitive), whereas JSP parameters are limited to Strings. In general you should always use `dsp:param` tags within a `dsp:include` tag.

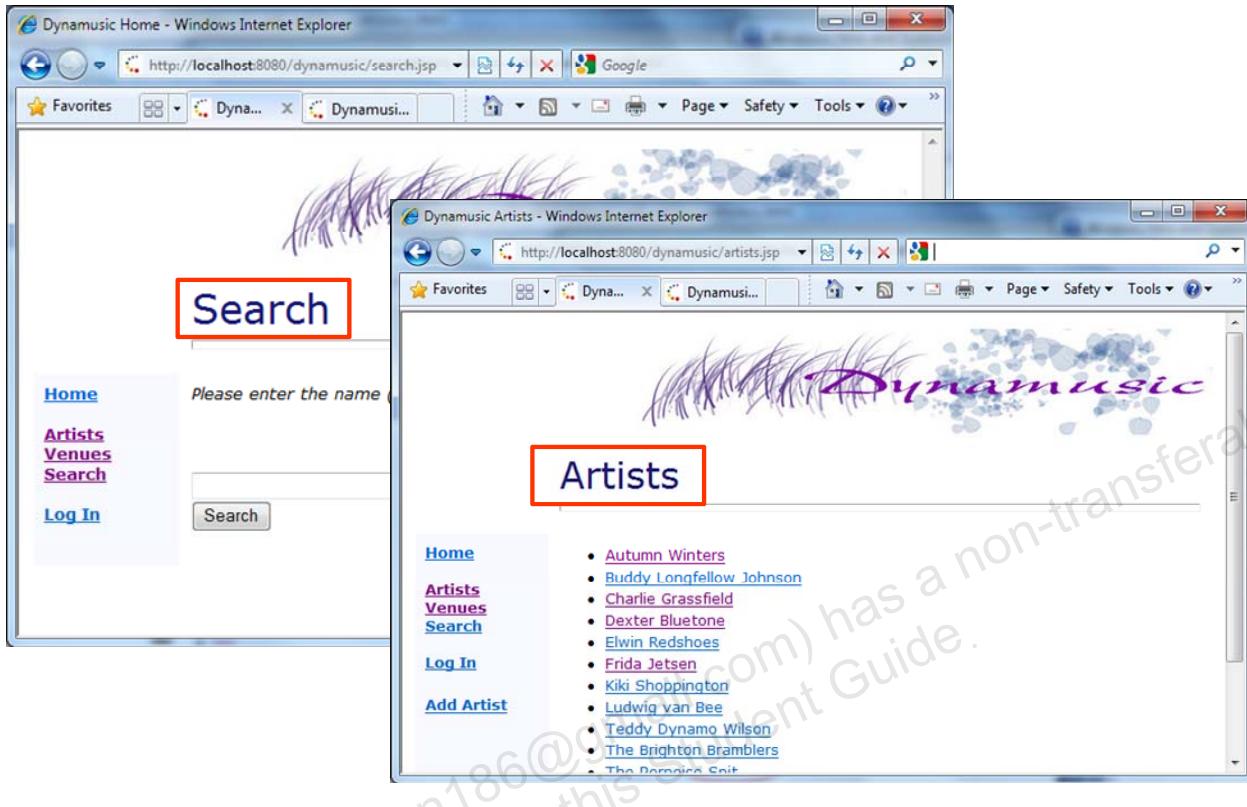
Page-level parameters can be thought of as “global” to the page, and parameters passed within the droplet tag (covered shortly) as “local” parameters to the embedded page. Parameters can be passed down but not back.

### `dsp:param`

The `dsp:param` tag actually has three possible attributes that can be used depending on the source of the value to be assigned:

- `<dsp:param name="param" value="...">`: Used when assigning a constant value or the result of a JSP expression, such as an object, to the parameter
- `<dsp:param name="param1" param="param2">`: Used to assign the value of `param2` to `param1`
- `<dsp:param name="param" bean="component.property">`: Used to assign the value of a component property to a parameter (covered later in the course)

## Example: Passing Parameters



ORACLE

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

In addition to their logo, Dynamusic would like to have a standardized way to display the "name" of the current page. This name varies for each page, but the font and placement is the same.

## Example: Passing Parameters

search.jsp

```
<%@ taglib uri="/dspTaglib" prefix="dsp" %>
<dsp:page>
<html>
  <body>
    <dsp:include page="common/header.jsp">
      <dsp:param name="p1" value="Search"/>
    </dsp:include>
    ... Please enter the name (or first few...
  </body>
</html>
</dsp:page>
```



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

In the example in the slide, a parameter called `p1` is generated when the page is rendered, and its value is set to “Search” (a literal, in this case).

## Example: Passing Parameters

common/header.jsp

```
<%@ taglib uri="/dspTaglib" prefix="dsp" %>
<dsp:page>
    
    ...
    <font color="midnightblue"...
        <dsp:valueof param="p1"/>
    </font>
    ...
</dsp:page>
```



ORACLE

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

Note that in the example in the previous slide, the “page fragment” did not require the `<dsp:page>` tag, because it did not contain any dynamic code. In the example in this slide, to retrieve the value of the `p1` parameter, the page must declare the DSP tag library and enclose the page in `<dsp:page>` tags.

In this example, if “`p1`” is not set, nothing will display. If they wanted to, Dynamusic could include a default page name, such as “Dynamusic,” in which case the `<dsp:valueof>` call would look like this:

```
<dsp:valueof param="p1">Dynamusic</dsp:valueof>
```

## Dynamic Parameter Values

Different syntax for parameters that have dynamic values:

- From a *component* property:

```
<dsp:param name="p1"  
bean="Profile.favoriteArtist.name"/>
```

- From another *parameter*:

```
<dsp:param name="p1"  
param="artistItem.name"/>
```



ORACLE

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

The example in the previous slide showed a fixed parameter value. A more common example involves passing a parameter whose value is derived from another dynamic source: Another parameter or a component property.

Note this is different from most other tags in the DSP tag library, such as the `<dsp:setvalue>` tag, where the syntax follows this pattern:

### Using a static string:

- `<dsp:tagname ... value="abc"/>`

### Using a bean property:

- `<dsp:tagname ... beanvalue="Component.property"/>`

### Setting to another param:

- `<dsp:tagname ... paramvalue="OtherParamName"/>`

## Passing Parameters Via Links

- Parameters can be passed to other pages via a hyperlink using a DSP anchor tag (`dsp:a`).

```
<dsp:a href="artistDetails.jsp">  
  <dsp:param name="artistName"  
    bean="Profile.favoriteArtist.name"/>  
  Click here to see details about <dsp:valueof  
    bean="Profile.favoriteArtist.name"/>.  
</dsp:a>
```

- This has the same effect as passing query values via a URL:

```
<A HREF="artistDetails.jsp?artistName=TravisB.">...
```



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

The `dsp:a` tag works very similarly to the standard HTML anchor tag, with the added ability to pass parameters using the `dsp:param` tag. Remember that the `dsp:param` tag allows you to specify either a literal value (using the `value` attribute) or a dynamic value (using either the `bean` or `param` attributes).

Defining with parameter tags is cleaner and can use expressions, component values (provided they return Strings), or other parameters as the source of the parameter value.

Passing via a URL permits only a string value. This is the only method available through HTTP.

The other difference between the `<dsp:a>` tag and the regular HTML `<a>` tag is different processing of relative and absolute target pages. Instead of just the usual `href` attribute, `dsp:a` also supplies a `page` attribute. An absolute page reference, such as `/foo.jsp`, is specified for the `page` attribute, which resolves relative to the current web application root, whereas the `href` attribute resolves it relative to the server-wide doc root.

# Quiz

Which tag is used to set the value of a parameter?

- a. <dsp:include>
- b. <dsp:param>
- c. <dsp:page>
- d. <dsp:valueof>

ORACLE

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

**Answer: b**

# Quiz

From which of the following can you set values of parameters?

- a. Component property
- b. Another parameter
- c. A JSP expression
- d. A literal value



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

**Answer:** a, b, c, d

## Road Map

- JavaServer page basics
- JSPs and ATG
- DSP component access
- DSP page parameters and fragments
- **Dynamo Servlet Beans**
- Nested DSP tag restrictions



ORACLE

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

## What Is a Dynamo Servlet Bean?

A Dynamo Servlet Bean (also known as a “droplet” or “DSB”) is a Nucleus component that:

- Performs some function, such as checking if a component property is empty
- Renders appropriate HTML based on the results



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

### Why use Dynamo Servlet Beans?

A DSB:

- Provides the ability to generate dynamic content in a page using a Java object
- Provides access to Java functionality from dynamic web pages while keeping Java and HTML code separate
- Combines JavaBean and servlet functionality

A servlet is Java code that runs within the server, enriching the server's functionality. It is typically used to generate dynamic content (HTML).

A droplet is a servlet. The advantages of droplets over conventional servlets include the following:

- You can use multiple DSBs in a single page.
- DSBs can be nested and can pass parameters to each other.

A DSB is also a JavaBean and, therefore, it can have properties that can be accessed or set from pages.

Throughout this course, the informal term “droplet” is used synonymously with the more formal “Dynamo Servlet Bean.” The ATG documentation most often uses the formal term.

## DSB: Simple Example

Dynamusic provides both a secure (HTTPS) and nonsecure method of payment and wants to display the correct URLs.

`https://secure.dynamusic.com/dynamusic/payment.jsp`

To purchase a subscription using our secure server, click [here](#)  
if that doesn't work for you, try [this](#) non-secure link

`http://www.dynamusic.com/dynamusic/payment.jsp`

ORACLE

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

## ProtocolChange Servlet Bean: Example

```
<b>To purchase a subscription using our secure  
server, click  
<a href="#">  
here</a>  
</b><br> if that doesn't work for you, try  
<a href="#">  
this</a> non-secure link
```

To purchase a subscription using our secure server, click [here](#)  
if that doesn't work for you, try [this](#) non-secure link

ORACLE

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

## ProtocolChange Servlet Bean: Example – Oparam

```
<dsp:droplet name="ProtocolChange">  
    <dsp:oparam name="output">  
        <b>To purchase a subscription using our secure  
        server, click  
        <a href="#" ... ">  
            here</a>  
        </b><br> if that doesn't work for you, try  
        <a href="#" ... ">  
            this</a> non-secure link  
    </dsp:oparam>  
</dsp:droplet>
```

**To purchase a subscription using our secure server, click [here](#)**  
if that doesn't work for you, try [this](#) non-secure link



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

(Not shown in the slide is the import of the servlet bean with its full path: `<dsp:importbean bean="/atg/dynamo/droplet/ProtocolChange">`)

Dynamo Servlet Beans (droplets) have three types of parameters (params):

1. **Open parameters (oparams):** JSP fragments that specify the HTML rendered by the droplet. They may include the value of the output parameter. They may also call other droplets or other DSP tags. In the example in the slide, the oparam is called `output`.

The next slide covers the second and third parameters.

## ProtocolChange Servlet Bean: Example – Input and Output Parameters

```
<dsp:droplet name="ProtocolChange"> [input param]
  <dsp:param name="inUrl" value="./payment.jsp"/>
  <dsp:oparam name="output">
    <b>To purchase a subscription using our secure
    server, click
    <a href=" here</a>
    <br> if that doesn't work for you, try
    <a href=" this</a> non-secure link
  </dsp:oparam>
</dsp:droplet>
```

The diagram illustrates the flow of parameters. The 'inUrl' parameter (highlighted in blue) is an input parameter. It points to a 'secureUrl' output parameter (highlighted in green), which in turn points to a 'here' link. A 'nonSecureUrl' output parameter (highlighted in green) also points to the same 'here' link.

**To purchase a subscription using our secure server, click [here](#)**  
if that doesn't work for you, try [this](#) non-secure link

ORACLE

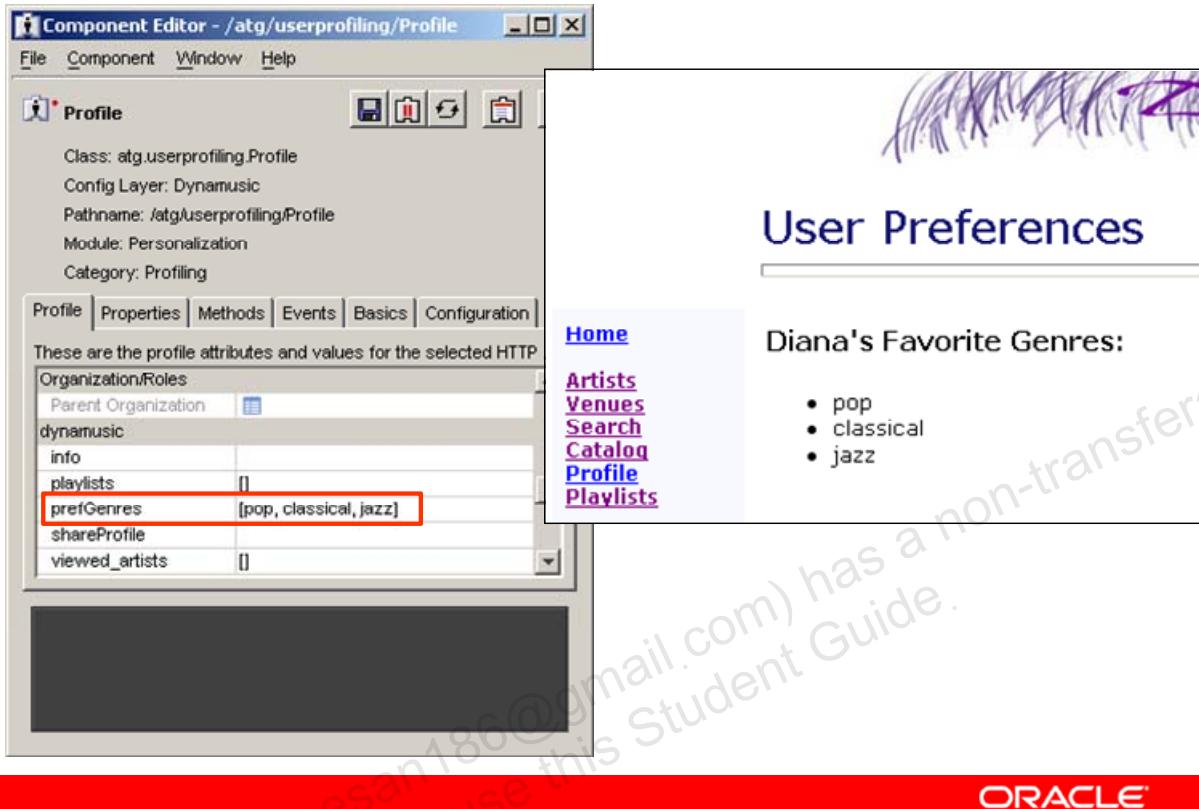
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

2. **Input parameters:** Values set by the page and passed into the droplet
3. **Output parameters:** Parameters set by the droplet, passed to and rendered within an oparam

The ProtocolChange droplet:

- Is primarily intended to allow for developers working in an insecure environment to develop code that is eventually deployed in an environment that provides both secure and nonsecure access
- Has properties that can be configured just like any other component. One of the properties is `ProtocolChange.enabled`, which defaults to `false` (in which case both the output params would have the same nonsecure value.) When the pages are deployed to a production environment, the system administrator can set that property `true`, and all pages will correctly render secure URLs. The example in the slide assumes that an administrator has set the `enabled` property on the droplet to `true`.

## ForEach Servlet Bean: Example



ORACLE

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

The `ForEach` droplet iterates over an unknown number of items in an “array.” In the example in the slide, the Dynamusic site shows a list of a logged-in user’s preferred genres, as captured in the `prefGenres` property of the `Profile` component.

## Using Dynamo Servlet Beans in a Page

```
<dsp:droplet name="/atg/dynamo/droplet/ForEach">
  <dsp:param name="array" bean="Profile.prefGenres"/>
  ...
  <!-- HTML to start the list -->
  <ul>
  ...
  <!-- HTML to generate for each item in the list -->
  <li> preference string
  ...
  <!-- HTML to finish the list -->
  </ul>
  ...
</dsp:droplet>
```

Diana's Favorite Genres:

- pop
- classical
- jazz

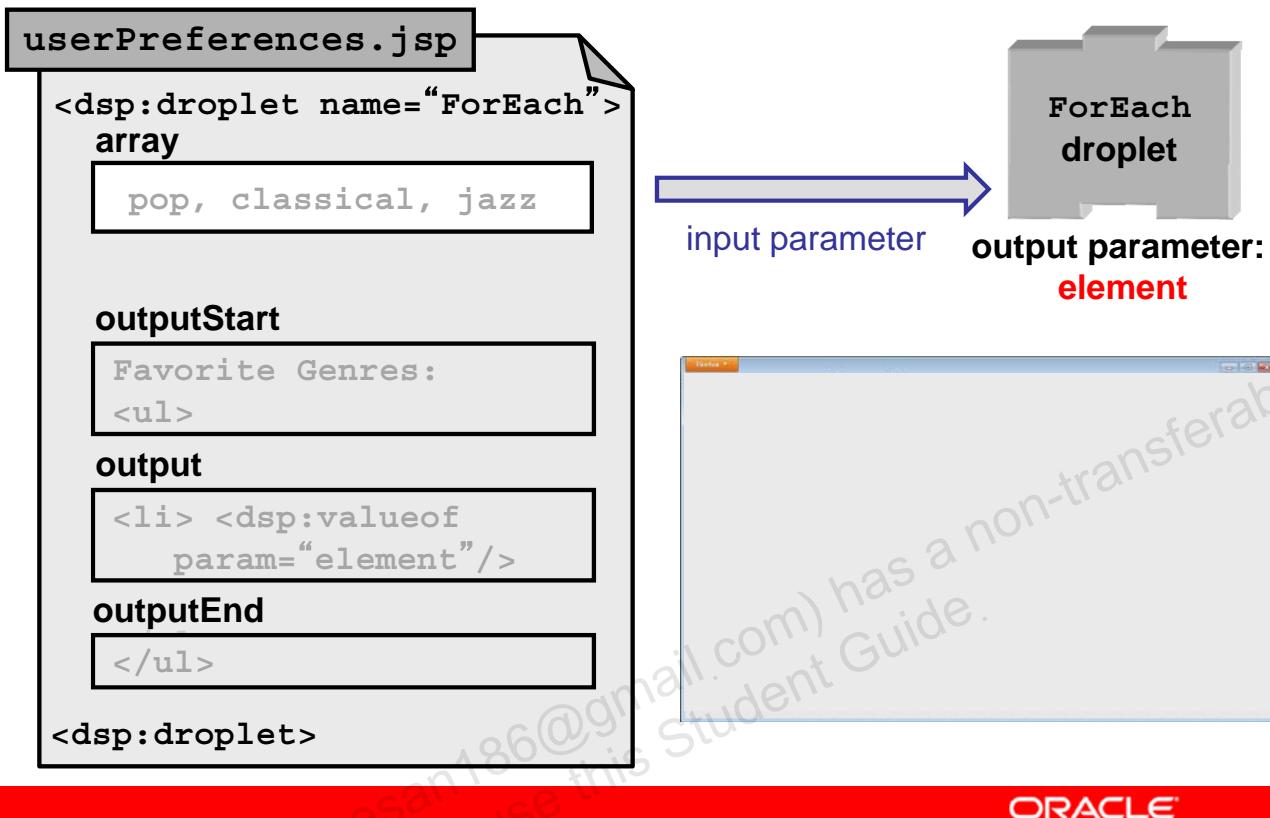


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The primary input parameter is called `array`. In the example in the slide, the `ForEach` droplet iterates over the `Profile.prefGenres` property, as defined by the input parameter. The droplet provides the same formatting for each item in the list, creating a list item using the standard HTML `<li>` tag.

**Note:** The `array` parameter can be a Collection (Vector, List, or Set), Enumeration, Iterator, Map, Dictionary, or array.

## Servlet Bean Parameters: ForEach



ORACLE

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

In addition to the parameters listed in the slide, `ForEach` also supports several other parameters. The example continues on the next few slides, showing the different parameters.

### Input Parameters

`sortProperties`: A string that specifies how to sort the list of items in the output array. Sorting can be performed on properties of JavaBeans, Dynamic Beans, or on Dates, Numbers, or Strings.

To sort JavaBeans, specify the value of `sortProperties` as a comma-separated list of property names. The first name specifies the primary sort, the second specifies the secondary sort, and so on. If the first character of each keyword is preceded by a `+`, this sort is performed in ascending order. If preceded by a `-`, it is performed in descending order.

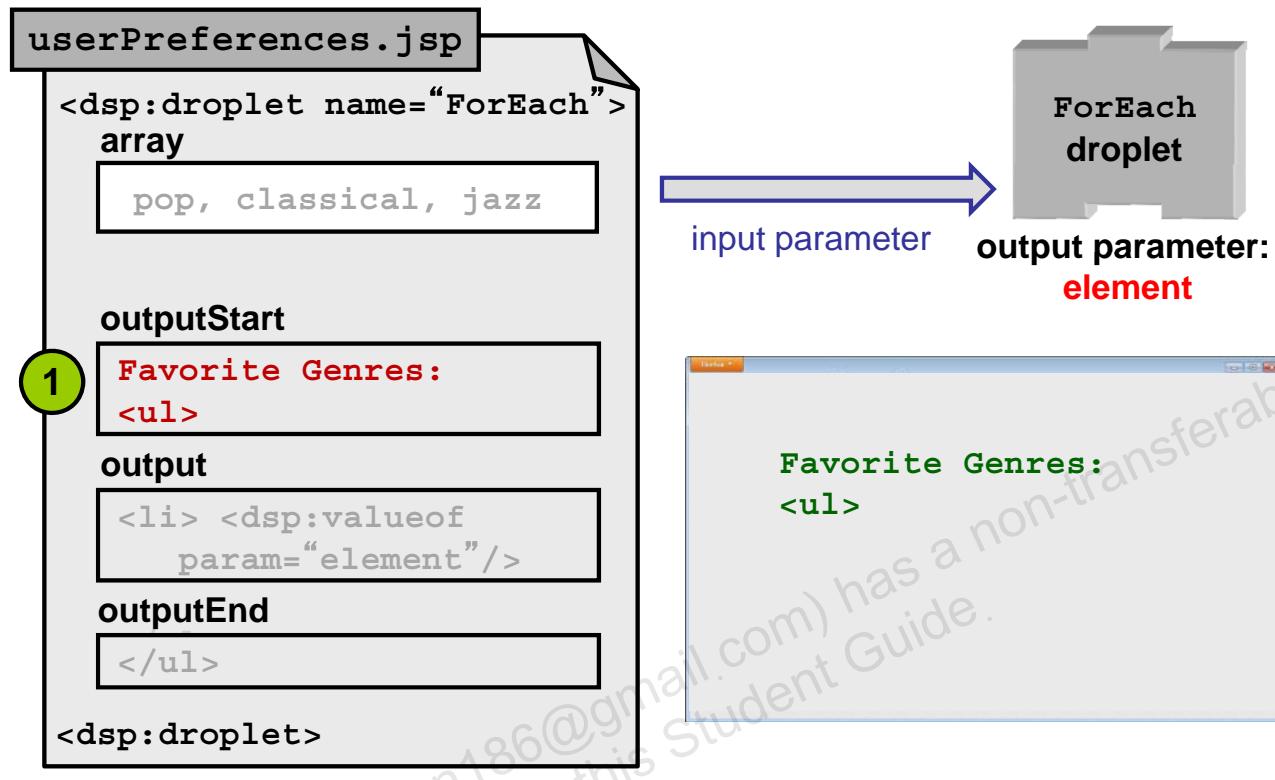
### Output Parameters

- `count`: This parameter is set to the one-based index of the current element of the array each time that the output parameter is rendered. The value of `count` for the first iteration is 1.
- `key`: If the array parameter is a Map or Dictionary, this parameter is set to the value of the key in the Map or Dictionary.
- `size`: This parameter is set to the size of the array, if applicable. If the array is an Enumeration or Iterator, `size` is set to -1.

### Open Parameters

- `empty`: Rendered if there are no objects in the input array

## Servlet Bean Parameters: ForEach

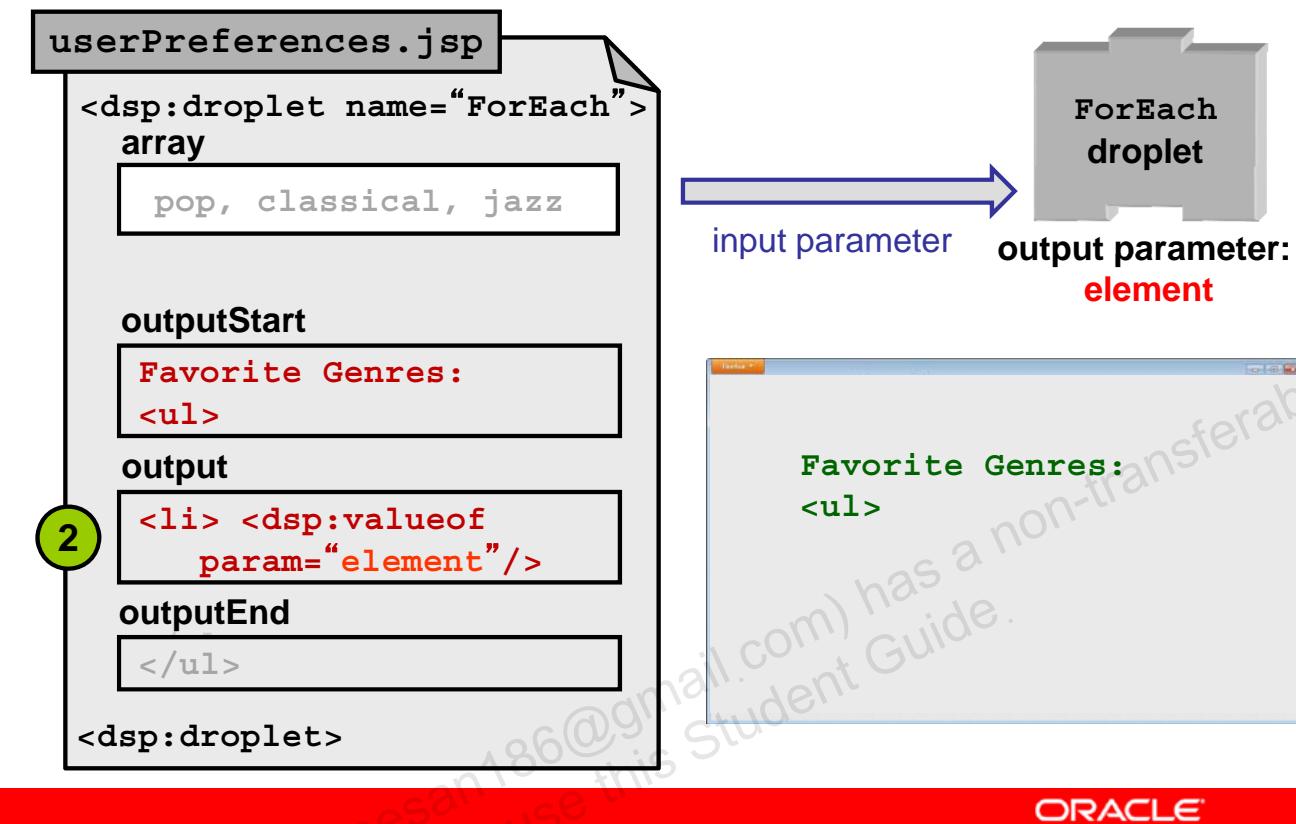


ORACLE

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

In the example in the slide, the `ForEach` droplet iterates over the profile's `preferredGenre` property. For the example, there are three genres to display: pop, classical, and jazz. `ForEach` requires the array as the input parameter. The `outputStart` oparam provides the title "Favorite Genres" and the HTML tag for an unordered list (`<ul>`).

## Servlet Bean Parameters: ForEach

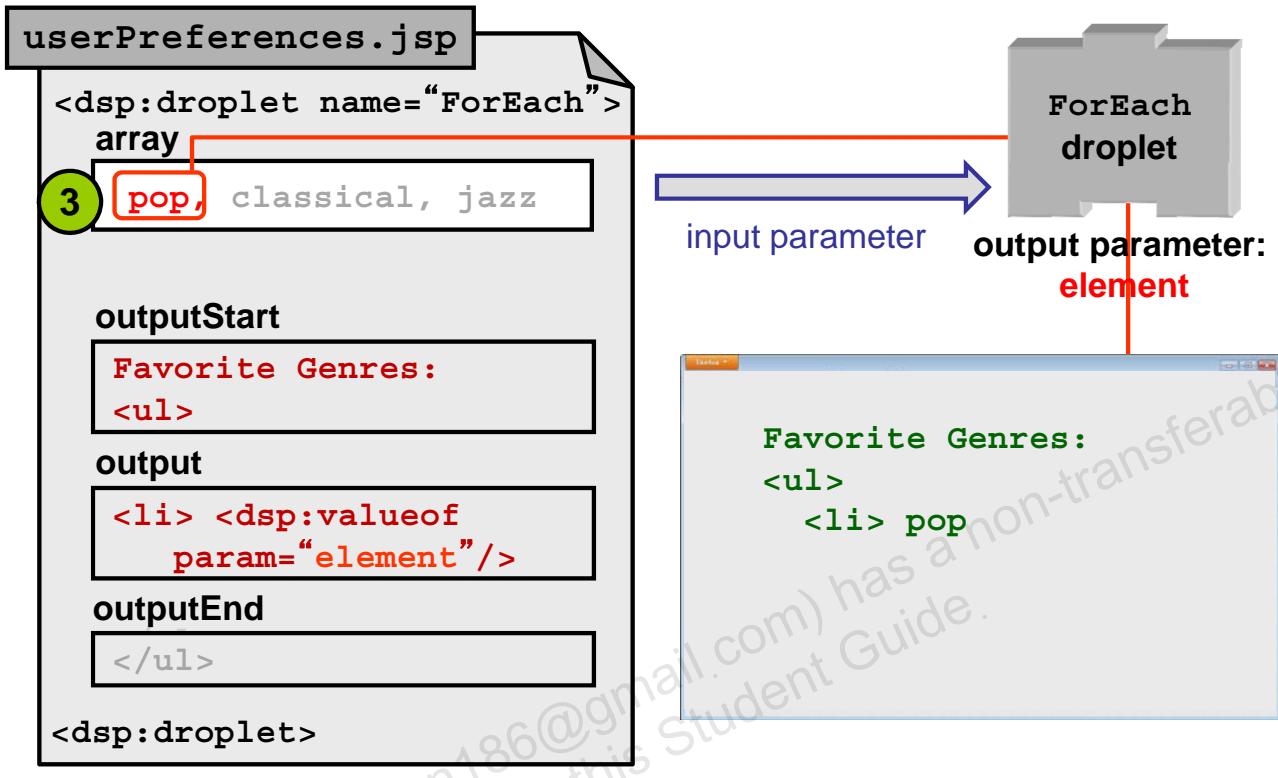


ORACLE

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

The output oparam lists each element of the array (each preferred genre in the example in the slide). `element` is the output parameter for the droplet.

## Servlet Bean Parameters: ForEach

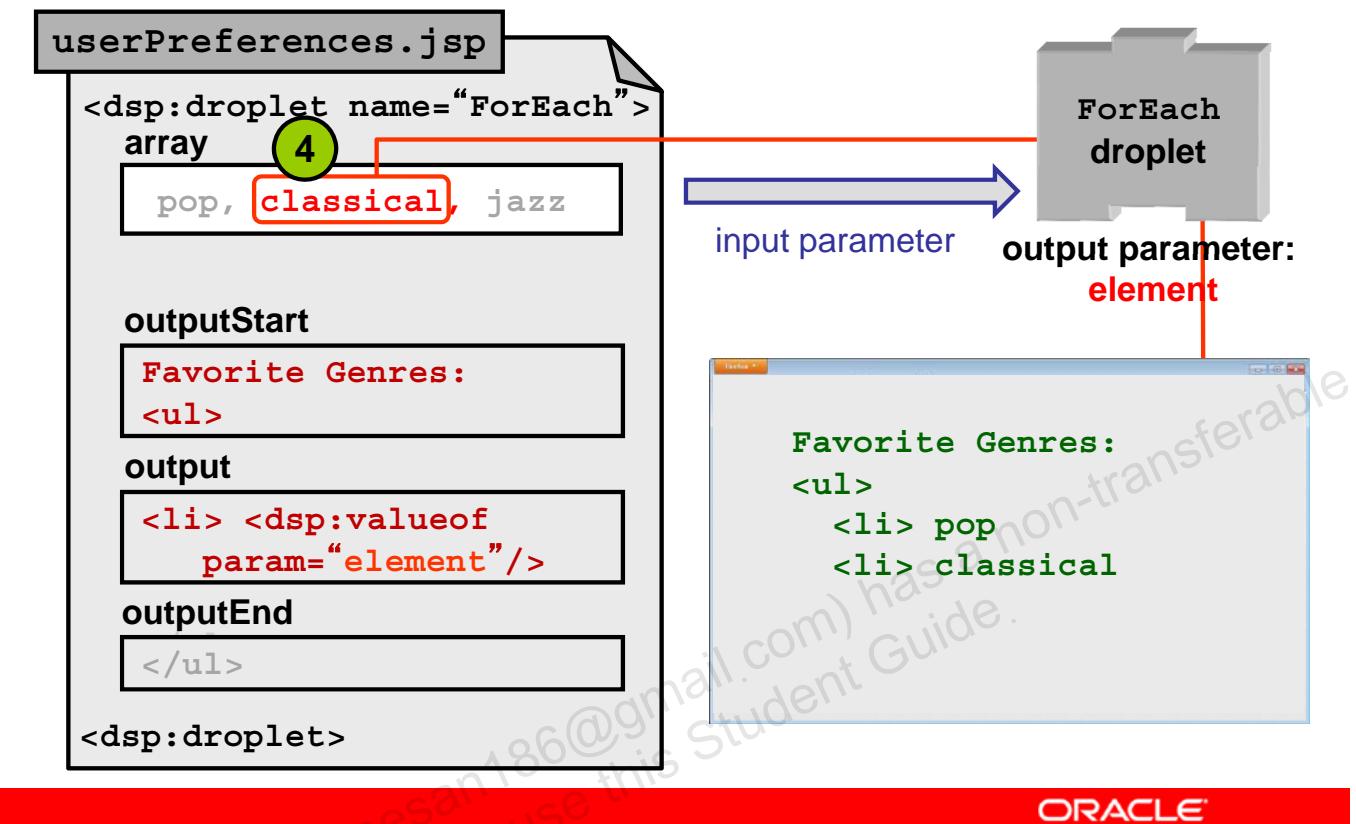


ORACLE

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

As the ForEach droplet iterates over the array, it renders the content of the output oparm for each element of the array. In the example in the slide, you see the list item “pop.”

## Servlet Bean Parameters: ForEach

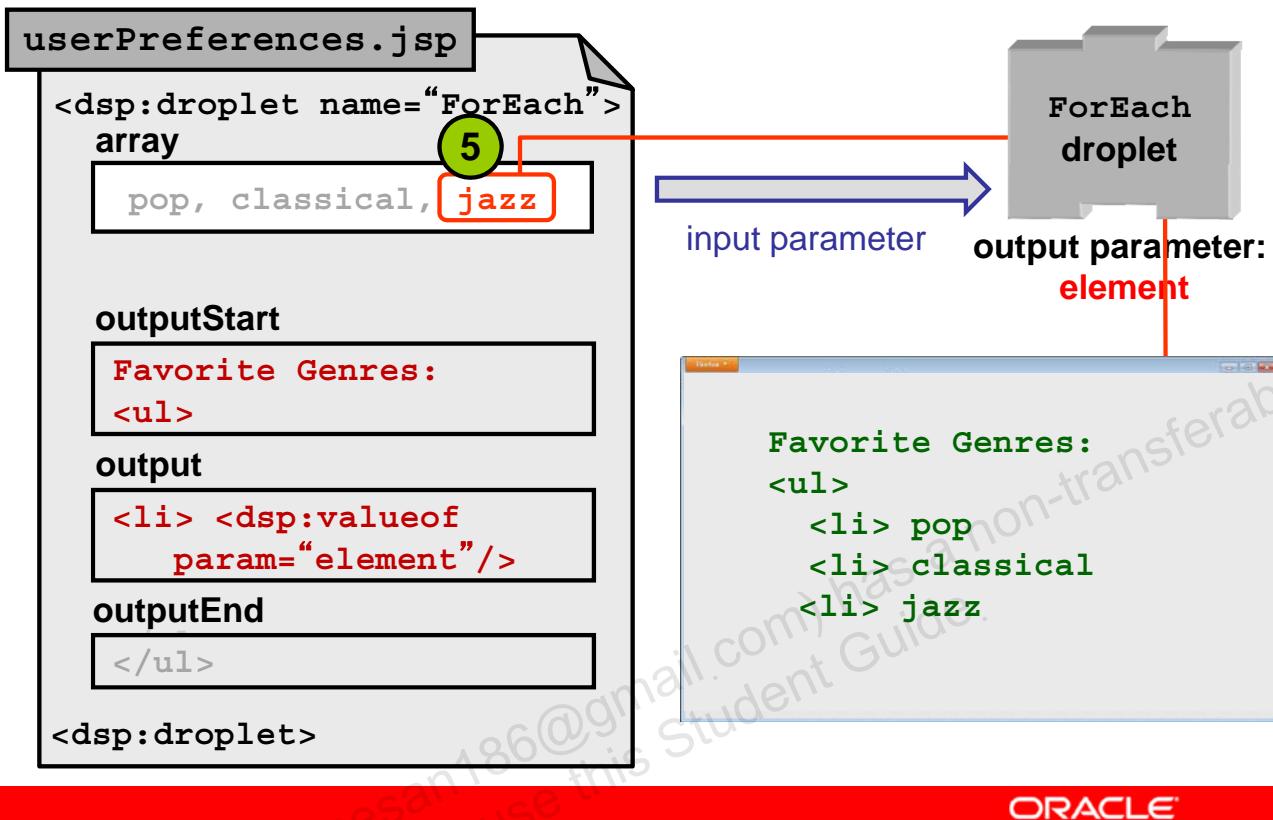


ORACLE

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

The ForEach droplet continues by listing the second item in the array, "classical."

## Servlet Bean Parameters: ForEach

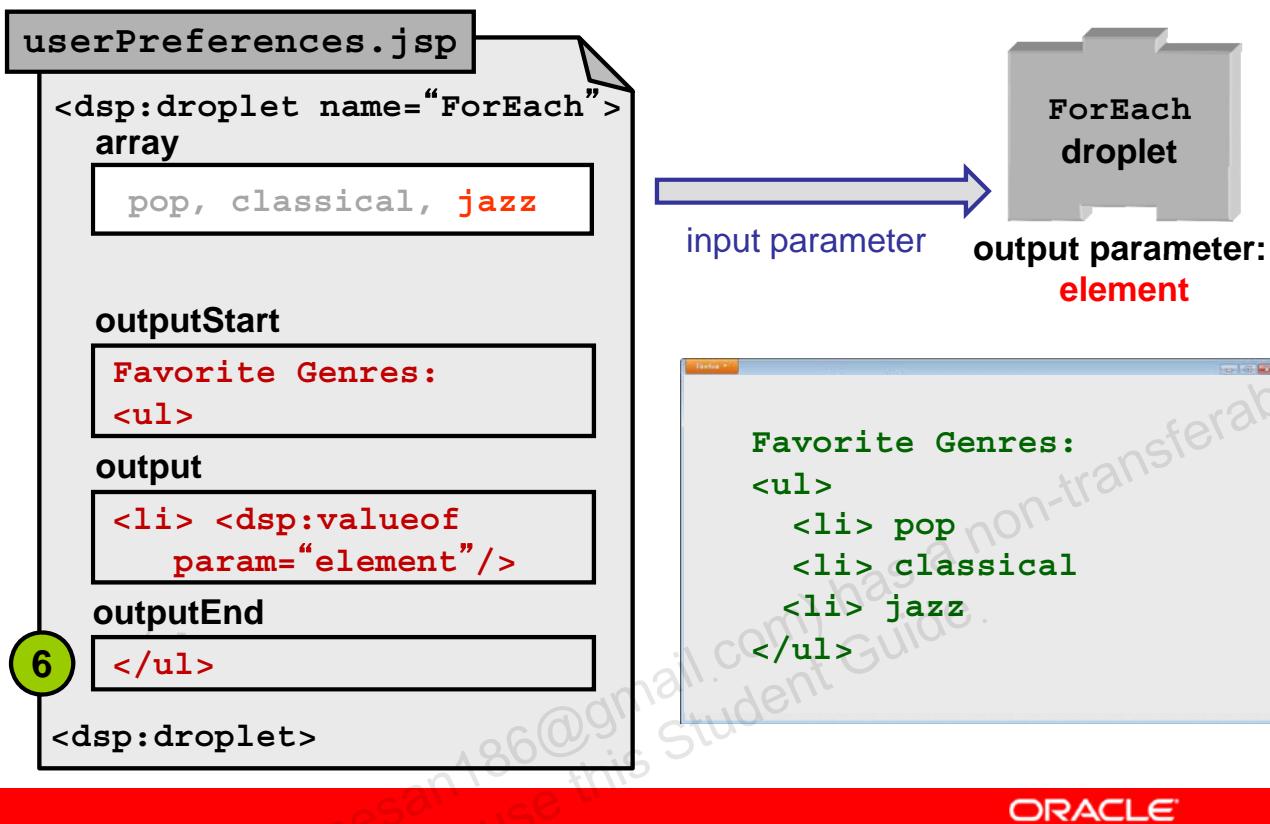


ORACLE

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

The third and final element in the array for this user is "jazz."

## Servlet Bean Parameters: ForEach



ORACLE

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

Once the droplet has iterated over all items, the `outputEnd` opparam is rendered. In the example in the slide, this opparam contains the closing tag for the unordered list.

## Example: ForEach Servlet Bean

The diagram illustrates the execution flow of an ATG ForEach droplet. It starts with an **input param** (highlighted in blue) which points to an **array** bean (highlighted in blue). This leads to an **outputStart** oparam (highlighted in purple), which generates an **ul** list item. The **output** oparam (highlighted in purple) then processes each item in the list, rendering an **element** (highlighted in green). If there are no items, it renders an **empty** message (highlighted in purple). Finally, the **outputEnd** oparam (highlighted in purple) closes the **ul** tag. The resulting output is a list titled "Diana's Favorite Genres:" containing three items: pop, classical, and jazz.

```
<dsp:droplet name="/atg/dynamo/droplet/ForEach">
    <dsp:param name="array" bean="Profile.prefGenres"/>
    <dsp:oparam name="outputStart">
        <ul>
    </dsp:oparam>
    <dsp:oparam name="output">
        <li> <dsp:valueof param="element"/>
    </dsp:oparam>
    <dsp:oparam name="empty">
        You have not specified any preferences.
    </dsp:oparam>
    <dsp:oparam name="outputEnd">
        </ul>
    </dsp:oparam>
</dsp:droplet>
```

ORACLE

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

The code shown in the slide is located in the course examples folder:

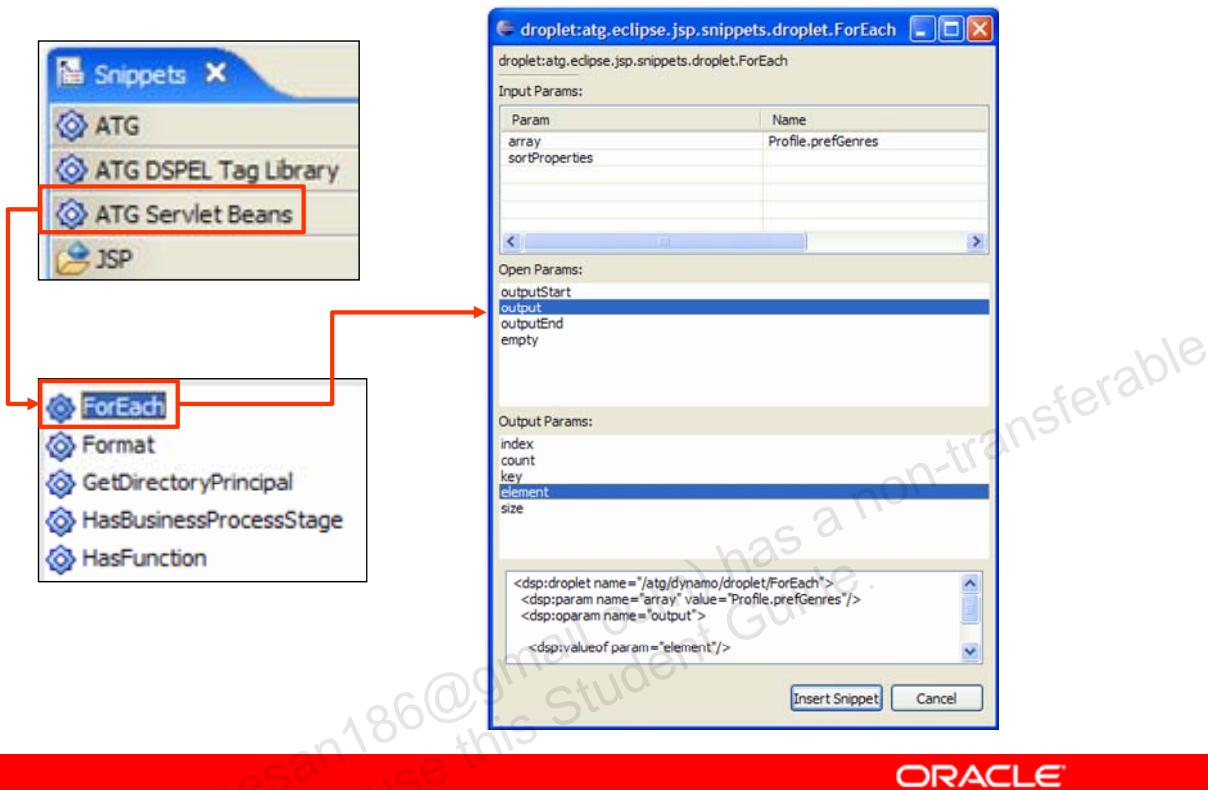
<coursedir>/examples/chapter03/listPrefs.jsp.

Input parameters can be defined within the droplet tag or in the page prior to the droplet tag.

Oparams are the basic mechanism for keeping Java and HTML separate. Use oparms to send the droplet any HTML that needs to be generated. The oparms specify the HTML (text and formatting) that can appear on the page whereas the Servlet Bean decides which of the oparms is rendered. Later slides discuss the third type of parameter: output parameter.

## Using Servlet Bean Snippets

In class, you may use Servlet Bean Snippets.



ORACLE

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

For your convenience, in this course you have been provided with several Eclipse "Snippets" (small configurable bits of code that can be dynamically added to pages) to help you with using droplets. These snippets enable you to select the appropriate droplet, and then configure the parameters.

To view available snippets in Eclipse, select Window > Show View > Other > Snippets. For more information about Snippets, see <http://www.eclipse.org/swt/snippets/>.

## Some Frequently Used Servlet Beans

- Dynamo Servlet Beans are used extensively in this course and in ATG reference applications, including:
  - ForEach
  - Switch
  - IsEmpty
  - IsNull
- Details about all droplets included in ATG can be found in the *ATG Page Developer's Guide*.



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

Most droplets follow the same pattern as the example `ForEach` covered in the previous slides, and are straightforward to learn from the documentation. They are not covered in detail here. You will want to familiarize yourself with the documentation for `Switch`, `IsEmpty`, and `IsNull` in particular because the exercises for this course make use of these droplets.

# Quiz

A Dynamo Servlet Bean \_\_\_\_\_.

- a. Can also be referred to as a droplet
- b. Outputs HTML
- c. Is written in JavaScript
- d. Provides the ability to generate dynamic content in a page using a Java object



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

**Answer:** a, b, d

# Quiz

The tag to display the resulting HTML of a droplet is:

- a. <dsp:param>
- b. <dsp:valueof>
- c. <dsp:oparam>
- d. <dsp:droplet>



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

**Answer: c**

## Road Map

- JavaServer page basics
- JSPs and ATG
- DSP component access
- DSP page parameters and fragments
- Dynamo Servlet Beans
- Nested DSP tag restrictions



ORACLE

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

## Nested DSP Tag Restrictions

DSP does not allow one DSP tag to be used as a dynamic attribute value for another DSP tag.

```
<dsp:droplet name="ProtocolChange">
  <dsp:param name="inUrl" value=".//payment.jsp"/>
  <dsp:oparam name="output">
    ...
    <dsp:a href="<dsp:valueof param='secureUrl' />">
      <dsp:param name="userid"
        beanvalue="Profile.id"/>
      here
    </dsp:a>
    ...
  </dsp:oparam>
</dsp:droplet>
```



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

Recall from a previous example that the ProtocolChange droplet has an output parameter called secureUrl. In that example, the parameter used by an HTML anchor tag was:

```
<a href="
```

This worked because the example used the `<a>` tag and not the `< dsp:a>` tag.

In that example, only a simple URL was required, so `<a>` worked fine. What if you needed to pass an ATG value as part of the link? Then you need to use a `< dsp:a>` tag instead, and unfortunately, you cannot nest a DSP tag within another. In that case, `< dsp:a href=" does not work.`

## Nested DSP Tag Restrictions

Use `dsp:getvalueof` to assign the value to a scripting variable.

```
<dsp:oparam name="output">
...
<dsp:getvalueof param="secureUrl" id="surl"
    idtype="java.lang.String">
    <dsp:a href="<% surl %>">
        <dsp:param name="userid"
            beanvalue="Profile.id"/>
        here
    </dsp:a>
</dsp:getvalueof>
...
...
```

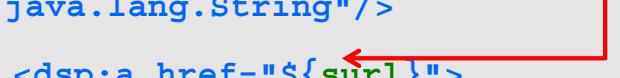
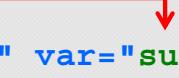


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To work around the limitation of nested tag attributes, use the `dsp:getvalueof` tag to assign the value you want to a local Java variable name, and then use a JSP Expression tag to access the value. Using the `id=` attribute generates a scripting variable in the JSP. The example in the slide assigns `surl` as the value for the `secureUrl` parameter. You then use `<%= surl %>` in the `dsp:a` tag to access the parameter.

## Using `dsp:valueof` with EL Variables

You can use `dsp:getvalueof` with Expression Language (EL) variables, with no need to enclose the block where the variable is used:

```
<dsp:oparam name="output">
...
<dsp:getvalueof param="secureUrl" var="surl"
    vartype="java.lang.String"/>
    <dsp:a href="#">
```

```

        <dsp:param name="userid"
            bean="Profile.id"/>
            here
        </dsp:a>
...

```



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

In the example in this slide, the `dsp:getvalueof` tag does not need to enclose the area where the output variable is being used, which can be helpful in terms of organizing the page content. Using the `var=` attribute populates an Expression Language (EL) variable that stays in scope outside the body of the `dsp:getvalueof` (you can specify the scope in the tag if you like).

Note that for this use, EL variables must be enabled in your web application. If you use the ATG plug-in in Eclipse to create your module, it will create a `web.xml` file that uses the Servlet 2.3 spec, in which EL is disabled by default. See

`<coursedir>/solutions/chapter03` in the training environment for a `web.xml` file that uses the Servlet 2.4 spec in which EL is enabled by default. Alternatively, you may add the following at the top of your page: `<%@ page isELIgnored="false" %>`

See the *Page Developer's Guide* for more details about the `getvalueof` tag.

## Summary

In this lesson, you should have learned how to:

- Create JavaServer Pages that use the DSP tag library to access ATG functionality
- Embed page fragments into pages and pass parameters between pages
- Use Dynamo Servlet Beans (droplets) to generate page output



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

## For More Information

### *Page Developer's Guide*

- “Creating Java Server Pages”
- “Using ATG Servlet Beans”
- Appendix A: “DSP Tag Libraries”
- Appendix B: “ATG Servlet Beans”



ORACLE

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

## Practice 3 Overview: Dynamic Pages: Part 2

This practice covers the following topics:

- Using the Switch Servlet Bean to detect if the transient property is false before displaying the greeting (Practice 3-3)
- Using the ForEach Servlet Bean to loop through the list of the set of featured songs (Practice 3-3)
- Modifying the Dynamusic home page to include common sidebar and header fragments, passing a page name to the header fragment (Practice 3-4)



ORACLE

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

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

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

# Using ATG Repositories



ORACLE®

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

# Objectives

After completing this lesson, you should be able to:

- Describe the Data Anywhere Architecture
- Use an ItemLookupDroplet to retrieve an item from a repository
- Use the Repository Query Language (RQL) to query a repository
- Use the ACC to browse and query a repository

# Road Map

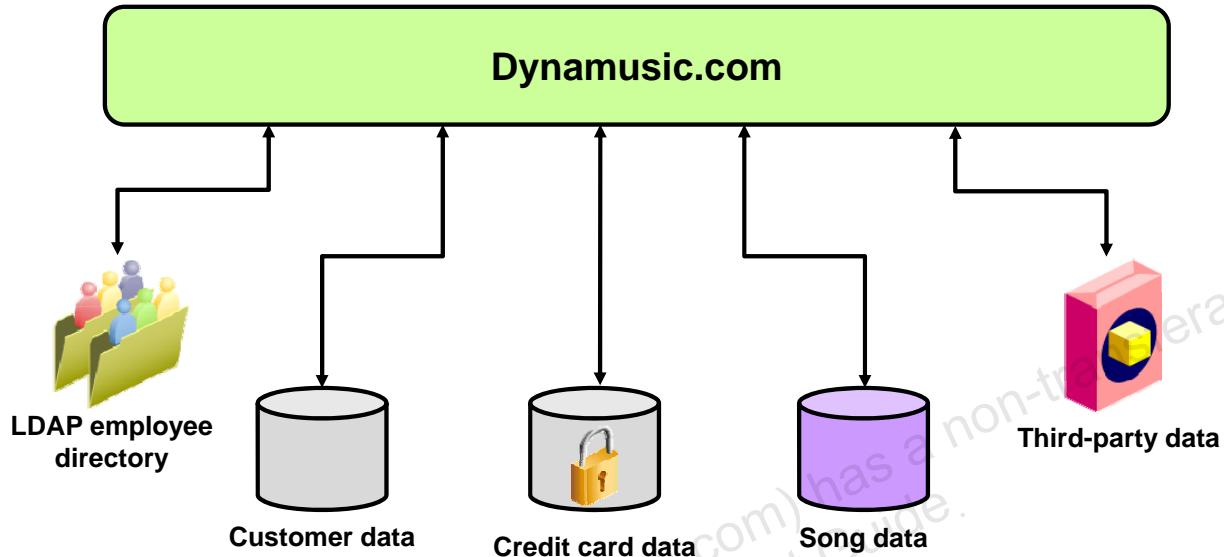
- ATG Data Anywhere Architecture
- Repository architecture
- Using repository servlet beans
- Querying a repository



ORACLE

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

## The Problem



ORACLE®

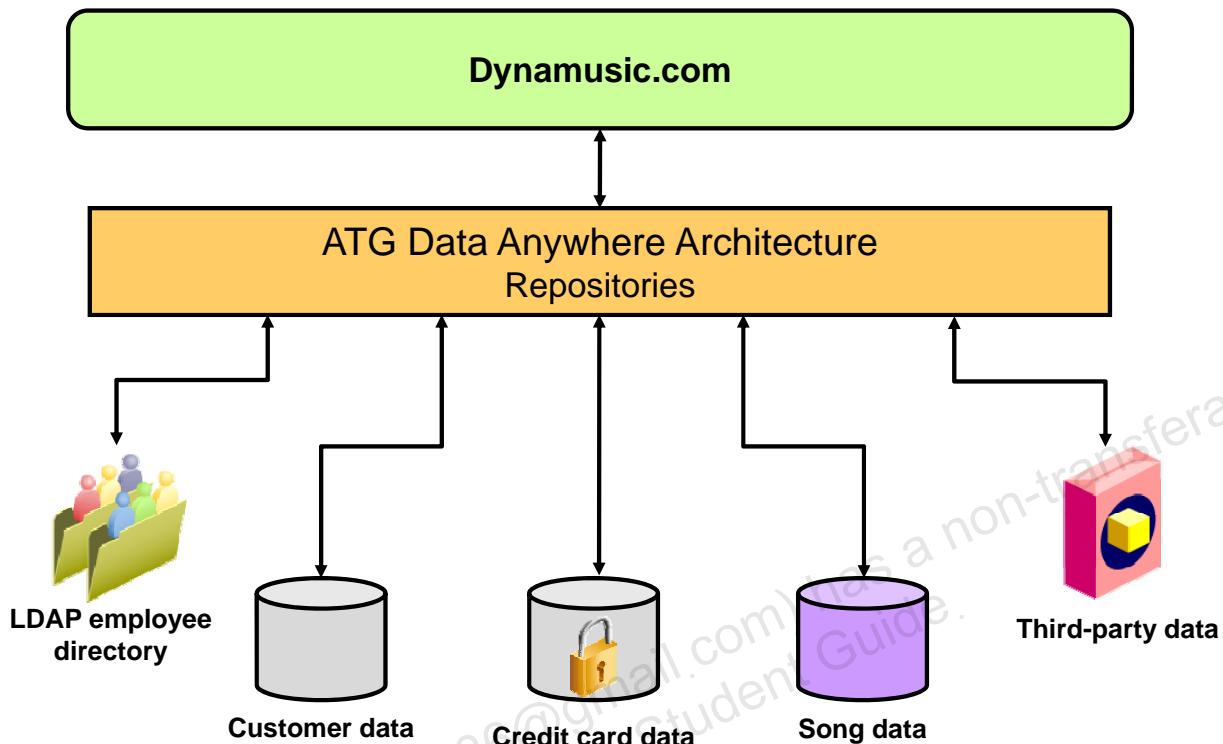
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Applications in today's world often have many sources and forms of data.

Imagine your hypothetical Dynamusic.com site. It may have employee information in an Lightweight Directory Access Protocol (LDAP) directory, client data in a database, music (provided by music publishing companies) in a different kind of database, and news (such as concert and album reviews from major news sources) that comes in from a web service. It may access data sources provided by third-party applications such as Hibernate.

All these different sources and kinds of data make data access in an application difficult, time consuming, and bug-prone.

# The Solution



ORACLE

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

The Data Anywhere Architecture converts all data, regardless of the source, into JavaBean objects (repository items), which can be easily accessed and manipulated in ATG applications.

Configuring a repository connection to a SQL database or an LDAP directory requires no Java coding at all.

This course focuses specifically on the most common type of repository: a SQL repository. But the principles covered apply to repositories that connect to LDAP and other data sources as well. Note that ATG supplies support for SQL and LDAP. You can add custom support for other data sources.

The relationship between ATG repository components and databases is not necessarily one-to-one. Administrators can choose to model several repositories from one schema of one database (which is what Dynamusic is doing—the Songs, Events, and Profile repositories are all accessing the same schema). Alternatively, a single repository item can be made up of data from multiple databases using a composite repository. In fact, these databases do not even need to be the same type: one could be SQL and one LDAP. ATG's repository layer does all the necessary assembling and splitting of queries and commands to the various data sources. For more on composite repositories, see the chapter titled "Composite Repositories" in the *Repository Guide*.

# ATG Repositories

ATG Data Anywhere Architecture repository infrastructure includes:

- ATG servlet beans for displaying data in a page
- Form handlers for modifying data in a page
- Java API for querying, loading, and storing data



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

This lesson focuses on the use of droplets (ATG servlet beans) to access data from a dynamic page.

The lesson titled “Using Repository Form Handlers” covers out-of-the-box form handlers. For information on custom components and custom form handlers using the repository API, attend the course titled *Extending the ATG Platform*.

## Where Repositories Are Used

- The ATG product suite uses repositories extensively.
- Some out-of-the-box repositories include:
  - Profile repository (SQL or LDAP)
  - Content repositories
  - Commerce repositories (Product Catalogs and so on)



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

This course focuses on SQL repositories in general, and specifically the profile repository and the custom repositories required by Dynamusic.com.

Content repositories provide storage for content items and the ability to associate metadata with content, allowing you to deliver targeted content. They can exist as either pure SQL repositories that store all of the content in tables or hybrid repositories that store references to file folders that contain the content.

Commerce repositories (such as the Catalog, and repositories containing orders, inventory information, gift certificates, and so on) are covered in the *Implementing Your ATG Commerce Solution* course.

# Road Map

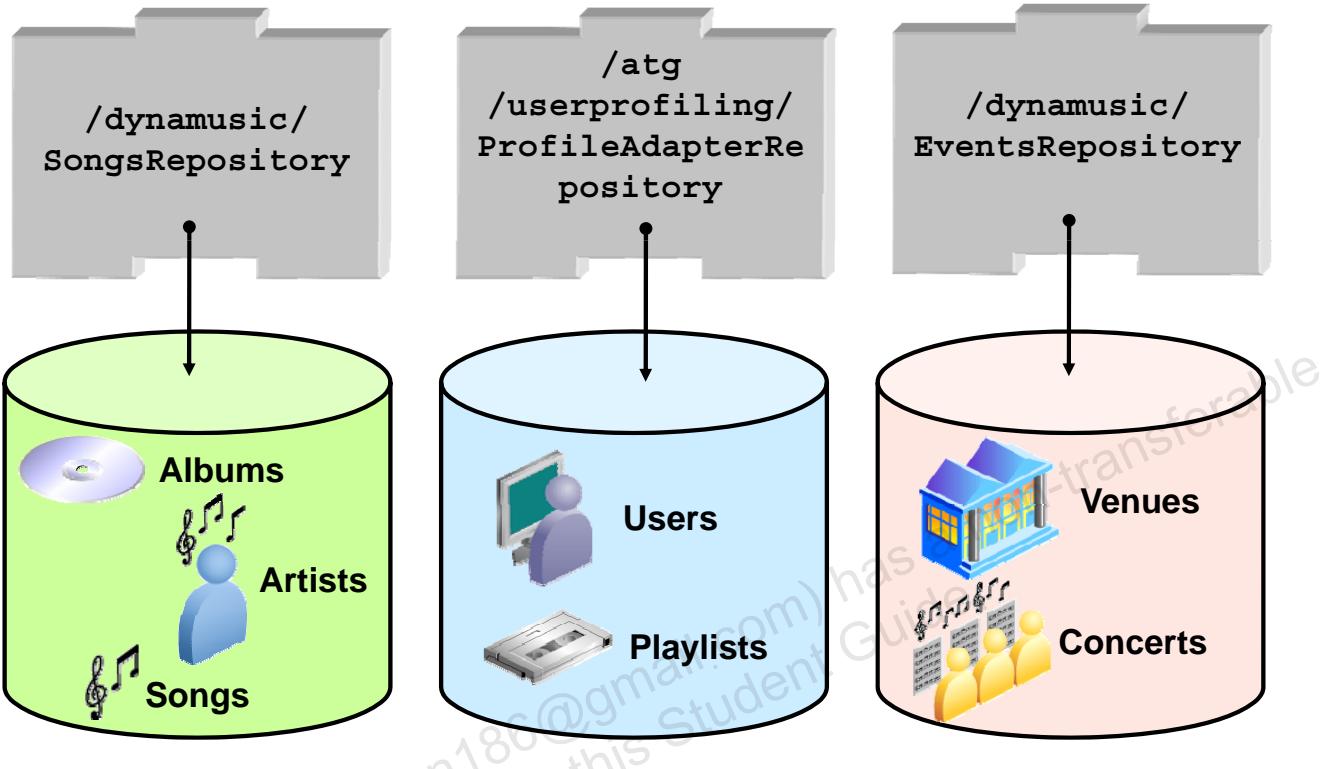
- ATG Data Anywhere Architecture
- Repository architecture
- Using repository servlet beans
- Querying a repository



ORACLE

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

## Dynamusic.com Repositories



ORACLE

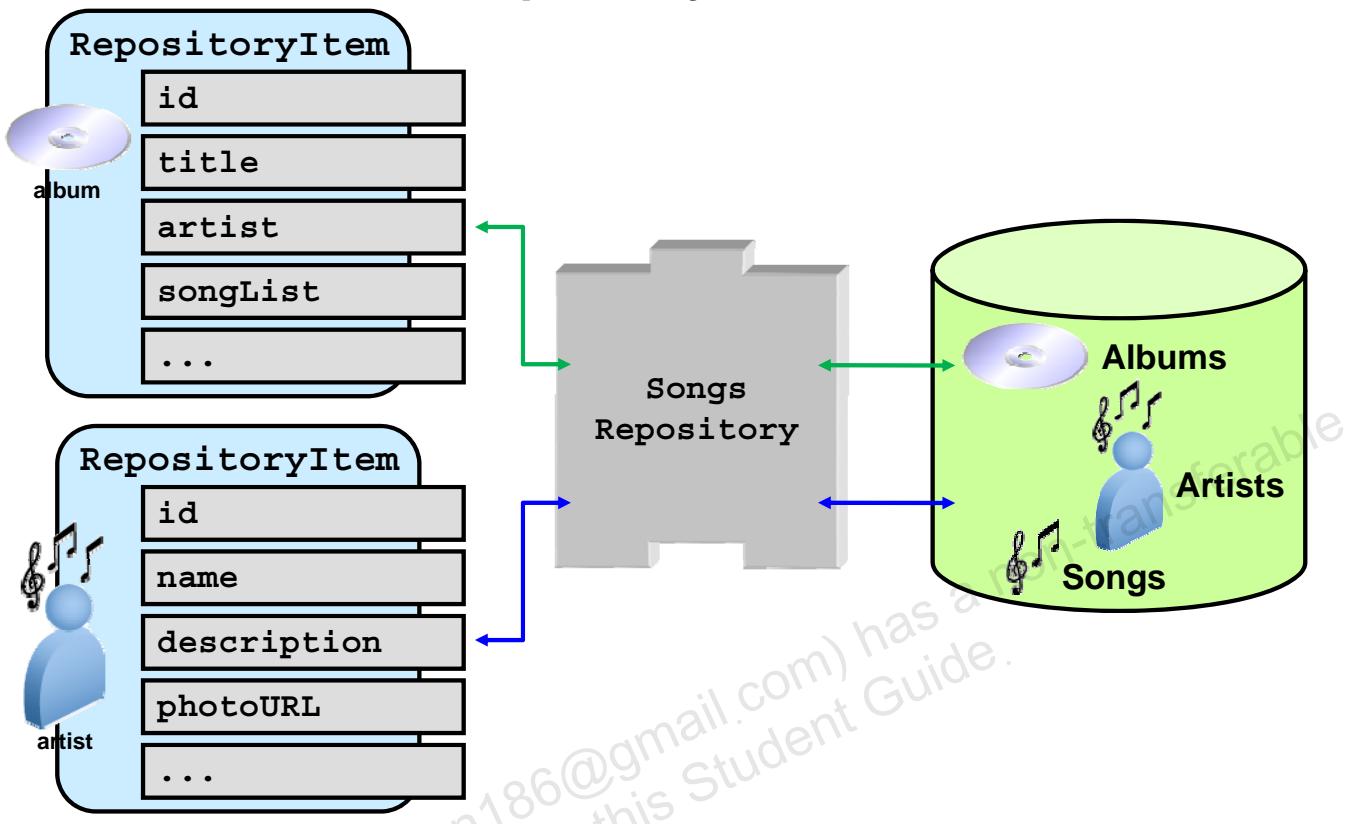
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In an ATG application, a *repository* is represented as a Nucleus component (of type `atg.repository.Repository`).

You can think of a repository as a collection of items (called “Repository Items”). Each item is an object that has an `id` property and other associated data properties.

Different repositories define different *item types*. For example, the `songs` repository might define item types `album`, `artist`, and `song` corresponding to the types of data stored in the data source (in the example in the slide, SQL databases).

# Repository Items



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

Each “unit” of data in the data source is mapped to an object of type `RepositoryItem` in ATG. Although a repository may contain any number of types of items, and there may be any number of repositories, every item will be returned by the repository API as an object of type `atg.repository.RepositoryItem`.

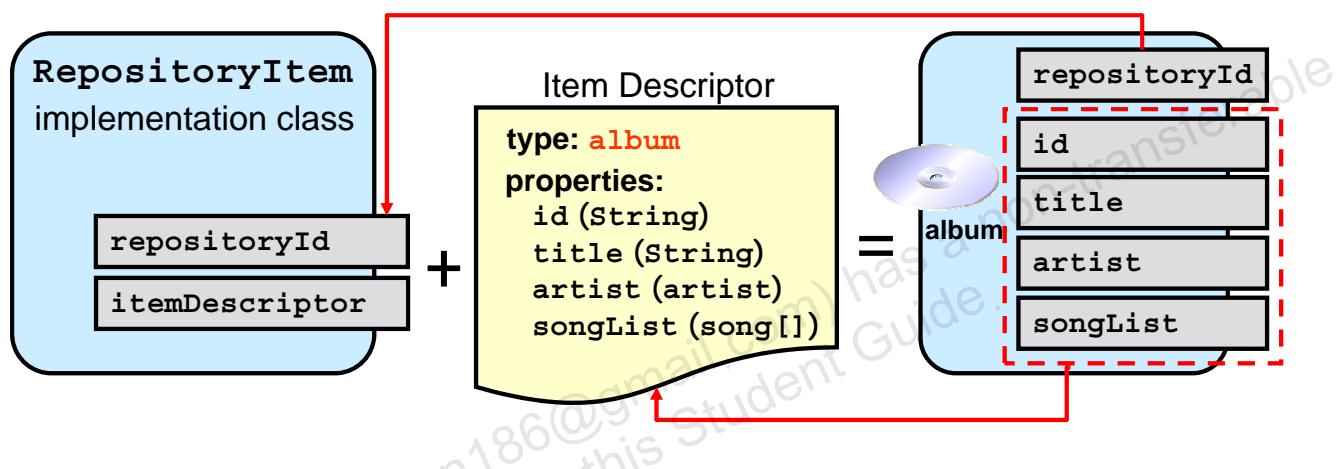
Repository items are categorized by *item type*. In the Songs repository, there are three item types defined: albums, artists, and songs. Each item in this repository is an instance of one of these types. The type determines what properties the associated `RepositoryItem` object has.

Are you wondering how a single Java Class (`RepositoryItem`) can have different properties on different instances? That is the magic of ATG Dynamic Beans, which store the special dynamic bean properties in a Map. Each element in the Map can represent a property, where the key is the property name.

# Repository Items

Each RepositoryItem has:

- A repositoryId property
- Zero or more other properties as specified by the Item Descriptor



**ORACLE**

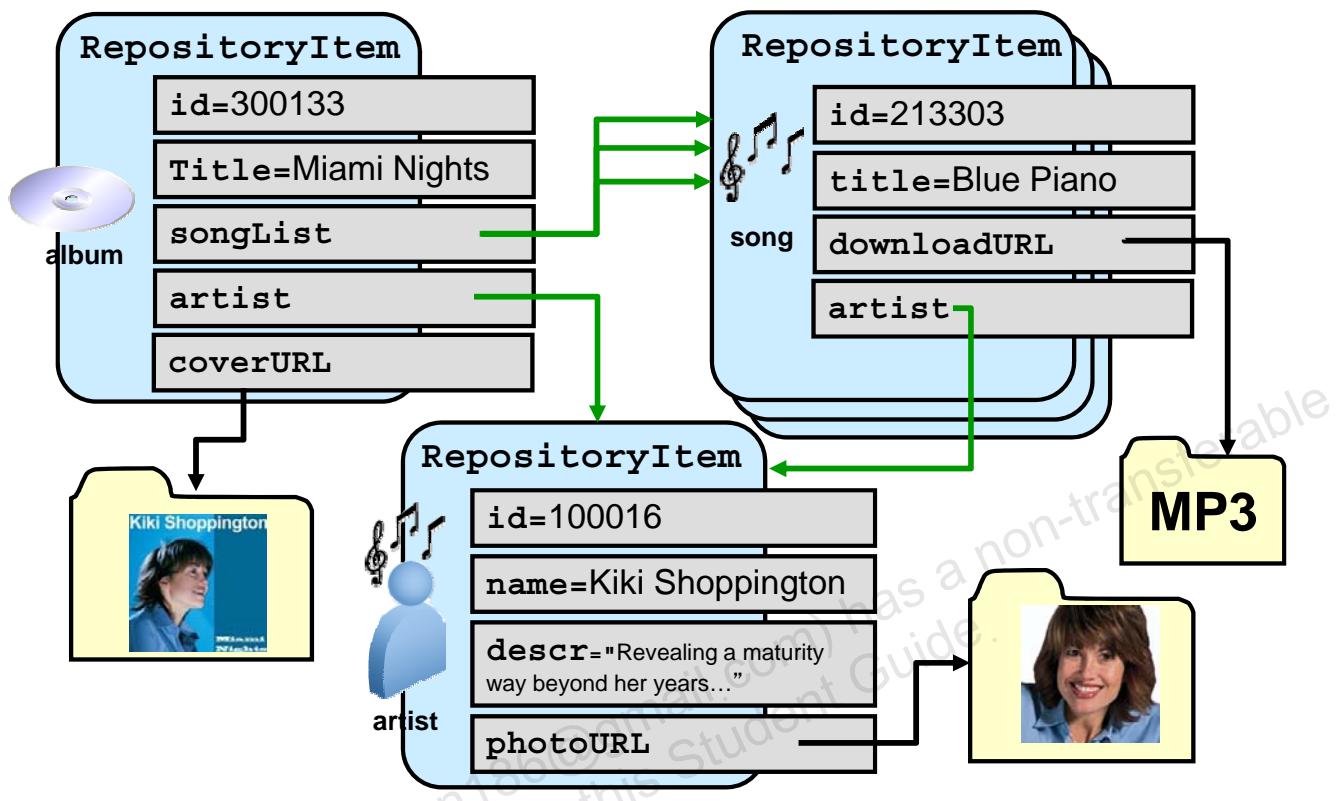
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Instances of `RepositoryItem` have a “genuine” bean property called `repositoryId` (meaning there is a getter and setter in the Java class). They also have dynamic bean properties that are configured in an XML file and stored in a Map on live instances. In the diagram in the slide, the properties inside the dashed box (`id`, `title`, `artist`, and `songList` from the `album` item type) are dynamic bean properties.

Note that, in the example, `album` has both an `id` property (as configured by the item descriptor) and a `repositoryId` property (because *all* repository items have that property, regardless of the definition of the item descriptor.) You will see later, when you understand how the `id` property is configured, that `id` is always the same as `repositoryId`. Why then, you might wonder, have an `id` property at all? It is not required. However, there are a few advantages to “exposing” the `id` as an item descriptor property:

- It makes it easy to determine the ID of an item you are viewing in the ACC (the `id` property shows up in the list of properties, the `repositoryId` property does not).
- You cannot build RQL queries (covered later in the lesson) against the `repositoryId` property. They can be built only against item descriptor properties.

## Item Relationships

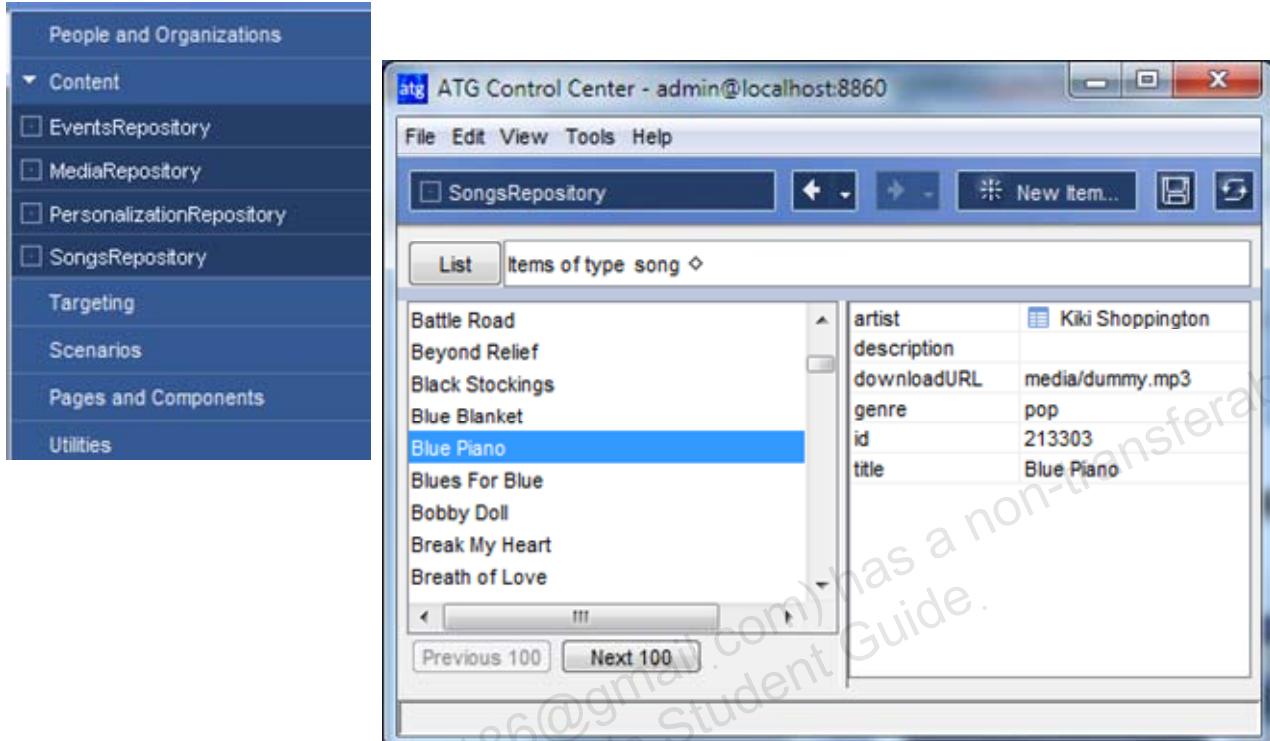


ORACLE

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

The example in the slide shows three item types: album, song, and artist. There are several ways that the repository could have been set up, and there are good arguments for each of them. For instance, the images (an album's coverURL or an artist's photoURL) could have been stored in the database (as “blobs”), or on the file system, with only file references being stored in the database. Also, Dynamusic could have defined a many-to-many relationship between artists and songs, but instead elected to have a pointer from song to artist, but not from artist to song. In part, this was chosen because this course only covers one-to-many relationships (leaving many-to-many for a follow-on course called *Extending the ATG Platform*) but this also makes sense given Dynamusic’s focus: the main item Dynamusic deals with is songs. The most common relationship that needs to be discovered, given a song, is who the artist is, so a link in that direction makes sense. Songs are also likely to be added and removed much more frequently than artists, and a song cannot exist without an artist, whereas the converse is not true.

# Demonstration: Repositories and the ACC



ORACLE

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

The instructor may demonstrate the following repository features in the ACC:

- Browsing items in the repository
- Constructing repository queries
- Adding a new item to the repository
- Linking repository items

# Road Map

- ATG Data Anywhere Architecture
- Repository architecture
- Using repository servlet beans
- Querying a repository

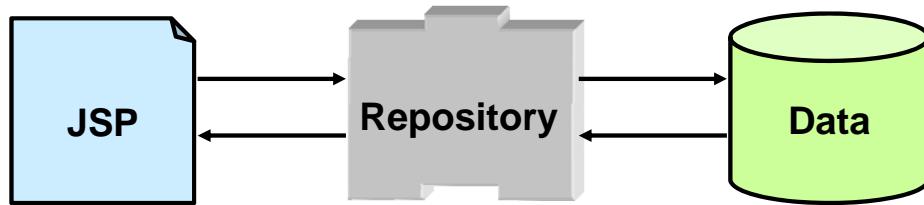


ORACLE

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

## Using Repositories with Dynamic Pages

- The most common use of repositories is to display and manipulate data in dynamic pages.



- ATG includes tools for using repositories in pages:
  - Servlet Beans
  - Form Handlers

ORACLE

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

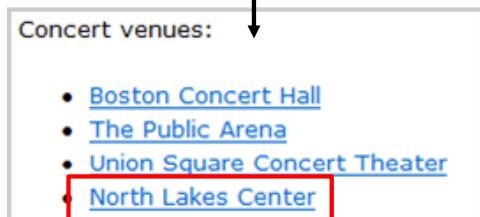
Servlet Beans are used to display repository contents. Form handlers help you create forms to allow users to manipulate data in a repository. This lesson covers Servlet Beans. Form Handlers are covered in the lesson titled “Using Repository Form Handlers.”

# Repository Servlet Beans

- Query for items

`RQLQueryForEach`

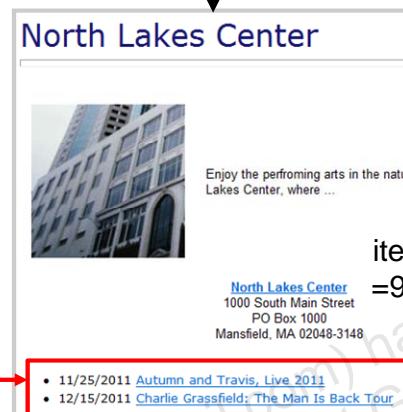
`RQLQueryRange`



`venues.jsp`

- Find an item by ID

`ItemLookupDroplet`



`venueDetails.jsp`



**ORACLE**

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

There are two types of droplets for accessing items in a repository: *query* droplets, which return all items meeting your specified criteria, and *lookup* droplets, which return a single item based on ID.

Queries are specified in “RQL,” which stands for Repository Query Language. The details of this language are covered shortly.

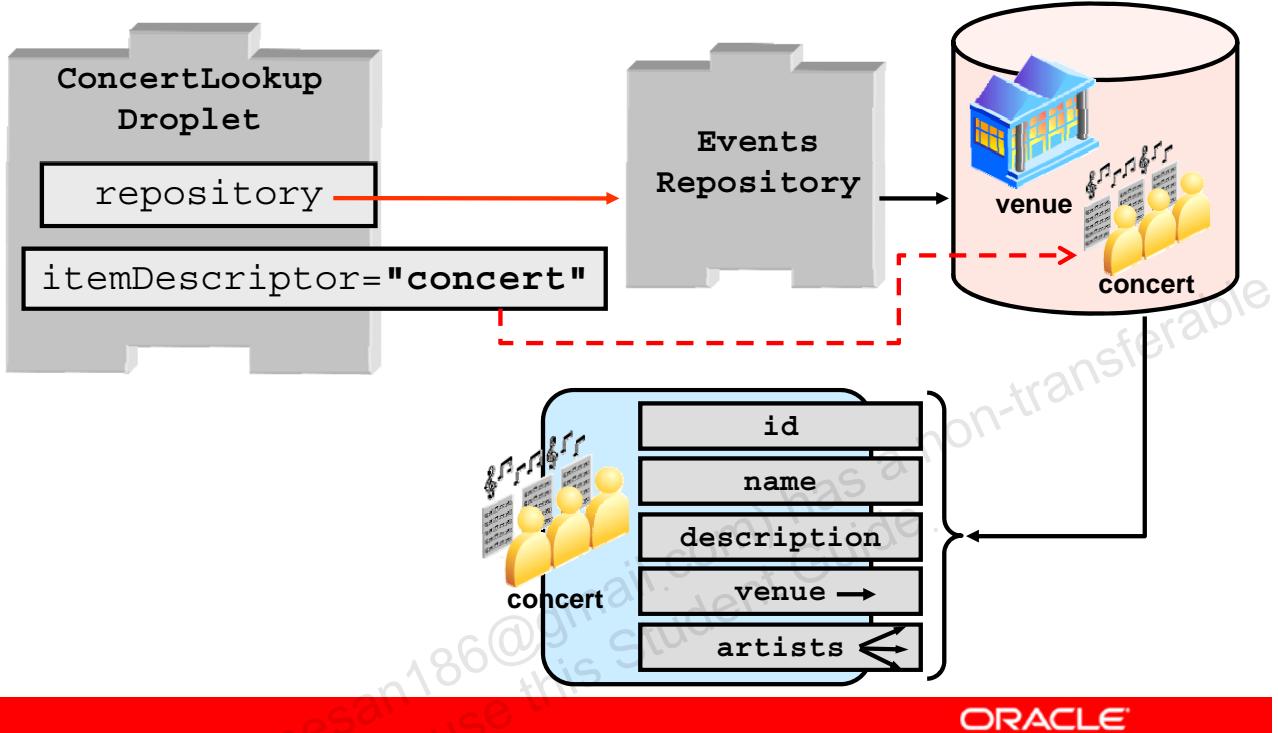
Query droplets and lookup droplets work together to create “drill-down” pages, where a query lists several items on the first page, and details of specific items are shown on subsequent pages using lookup.

An alternative to `ItemLookup` is `RepositoryLookup`. The main differences between these droplets is that `ItemLookupDroplet` can be configured with multiple repositories using a key, which is helpful for some approaches to internationalization.

In most cases, an `ItemLookupDroplet` is the preferred way to retrieve repository items, so that is what this course covers.

## Using an Item Lookup Droplet

1. Create and configure a new component based on the `ItemLookupDroplet` class.



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

The full path name of the class is `atg.repository.servlet.ItemLookupDroplet`. Note that each component based on the `ItemLookupDroplet` class is configured with a specific repository and itemDescriptor. It is possible to use a single `ItemLookupDroplet`, and pass in the repository and itemDescriptor as parameters instead of configuring them (by setting `useParams=true`), but creating separate components has better performance.

Another option is the use of the `RepositoryLookup` droplet, which is designed to have the repository and item descriptor passed in, rather than to have separate components for each item type.

## Using an Item Lookup Droplet

- Call from a page, passing in the `id` parameter.

```
<dsp:droplet name="/dynamusic/ConcertLookupDroplet">
  <dsp:param name="id" param="itemId"/>
  <dsp:oparam name="output">
    Name: <dsp:valueof param="element.name"/> <br>
    Venue: <dsp:valueof param="element.venue.name"/>
           <br>
    Date: <dsp:valueof param="element.date"
                      date="MMM dd, yyyy"/> <br>
  </dsp:oparam>
  <dsp:oparam name="empty">
    No such concert found.
  </dsp:oparam>
</dsp:droplet>
```

Name: Autumn and Travis, Live 2011  
Venue: North Lakes Center  
Date: Nov 25,2011



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

ItemLookup servlet beans are useful when you have an item ID from a repository and you need to display the item's properties. Where does the ID come from? Probably as the result of a query, discussed shortly.

# Quiz

A repository can access only database resources.

- a. True
- b. False



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

**Answer: b**

# Quiz

A repositoryID is an optional property for every repository.

- a. True
- b. False



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

**Answer: b**

## Road Map

- ATG Data Anywhere Architecture
- Repository architecture
- Using repository servlet beans
- Querying a repository



ORACLE

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

## Repository Query Servlet Beans

- To query for items
  - RQLQueryForEach: Displays each item found by a query
  - RQLQueryRange: Displays a subset of result items
- Queries can:
  - Match patterns in text
  - Query through collections
  - Query through complex values
  - Specify the order to present the results



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

The RQL query droplets are in the /atg/dynamo/droplet/ namespace.

# Using RQL Servlet Beans

## Input parameters

- repository: The repository to query
- itemDescriptor: Which item type to query for
- queryRQL: A statement written in RQL specifying the query criteria

## Output parameter

- element: The current item from the result set

## Oparam

- output: Rendered after the query has been executed
- empty: Rendered if no items were found that matched



ORACLE

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

# Repository Query Language (RQL)

Textual query syntax, similar to SQL

- RQL describes conditions to match items of a given item type.
- Why RQL?
  - Not specific to databases, can be used with any supported data source
  - Understands relationships between repository items
  - Higher level than SQL



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

The reason for using RQL instead of SQL is that Repositories are designed to support more than just relational databases. Remember that ATG repositories might be connected to relational databases, LDAP directories, or even file systems. Therefore, having a higher-level query language (such as RQL) is quite useful.

## RQL Statements

- RQL statements are built using the RQL syntax around a *specified item type descriptor*.
- Example:
  - Item type: song
  - Repository: /dynamusic/SongsRepository
  - RQL statement: genre = "jazz"  
Means “find all song items whose genre property is jazz”

**ORACLE**

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

Note that RQL is case-sensitive, including the operators.

## RQL Query Conditions

- Return all items: **ALL**
- Comparison: **=, !=, <, <=, >, >=**
  - Example: `age > 18`
- Text comparison: **STARTS WITH, ENDS WITH, CONTAINS**
  - Example: `firstName STARTS WITH "H"`
- **IS NULL queries**
  - Example: `phoneNumber IS NULL`



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

When comparison operators (**=, !=, <, <=, >, >=**) are applied to strings, it results in a lexical comparison.

The examples in the slide are querying against the `user` item type in the profile repository.

- `ALL` returns all items of type “user”.
- `age > 18` finds all users whose age property is greater than 18.
- `firstName STARTS WITH "H"` finds all users whose `firstName` property starts with “H” (use `IGNORECASE` for not case-sensitive comparison).

## RQL Queries with Item References

If a property refers to another repository item, you can drill down:

- Single value properties

```
address.zip STARTS WITH "02"
```

- Multivalue properties

```
favoriteArtists INCLUDES ITEM (name STARTS  
WITH "B")
```



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

- address.zip starts with "02" finds all users whose address property refers to an address repository item whose zip property starts with "02".
- favoriteArtists INCLUDES ITEM (name STARTS WITH "B") allows you to search the favoriteArtists property, which is a set referring to any number of artist repository items. This query finds all users whose favoriteArtists list includes at least one artist item with the name property beginning with "B".

## RQL Collection Queries

- INCLUDES ANY

For example:

```
prefGenres INCLUDES ANY { "jazz", "blues" }
```

is equivalent to:

```
(prefGenres INCLUDES "jazz") OR  
(prefGenres INCLUDES "blues")
```

- INCLUDES ALL

For example:

```
prefGenres INCLUDES ALL { "jazz", "blues" }
```

is equivalent to:

```
(prefGenres INCLUDES "jazz") AND  
(prefGenres INCLUDES "blues")
```



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

There are many other RQL operators. The details about RQL can be found in the *Repository Guide* in the chapter titled “Repository Queries.” Operators include:

- ORDER BY: Controls order of returned data set
- RANGE: Limits how many items are returned
- COUNT: Finds collections with a certain number of items
- MATCH/MATCHES: Uses database search facilities
- ID: Finds items with specific IDs

## Parameterized RQL Queries

- Queries have access to page parameters.
  - `genre = :paramname`
- The parameters you want to use must be in scope when the Servlet Bean is rendered.
- Parameters may be:
  - Passed in from an enclosing page
  - Defined on this page explicitly
  - Provided by the user by filling out a form

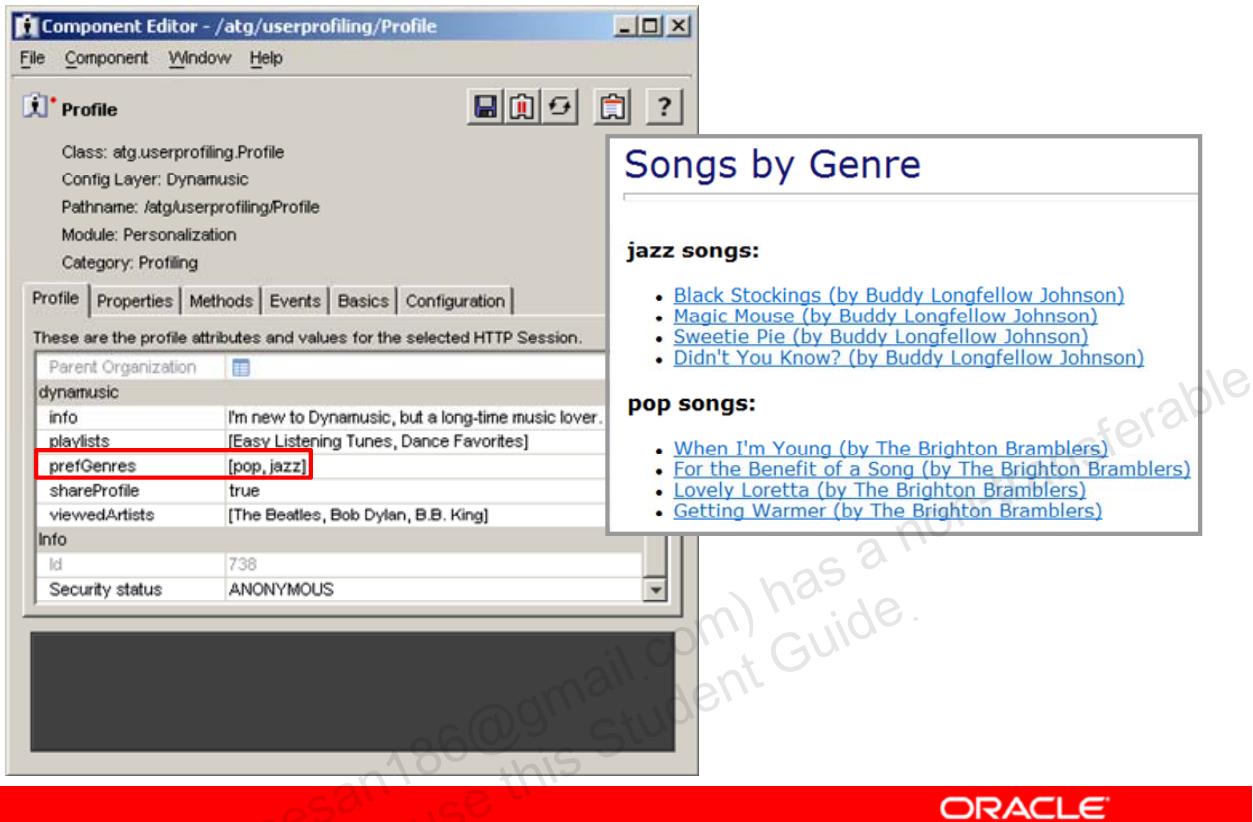


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

RQL is used in two different places: in RQL Servlet Beans, as discussed here, and in the repository API, discussed in the follow-on training course *Extending the ATG Platform*. The syntax for parameterized queries is slightly different when the API is used. Instead of `:paramname`, use `?0`, `?1`, `?2`, and so on, and the parameters are substituted in the order provided.

Note that you cannot parameterize property names (such as `genre` in the slide example). Only the values to search for (for example, `jazz`, `blues`, or `pop`) can be parameterized.

## RQLQueryRange: Example



ORACLE

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

To encourage browsing, Dynamusic wants to give users who have registered genre preferences a place to start browsing from. One way to do this is to provide a list of songs with genres that match the user's interests. In the example in the slide, the user Raoul has indicated an interest in pop and jazz. Dynamusic uses RQLQueryRange to find the first four songs that match that genre preference. (This is probably better done with a targeter, as covered in a different training course called *ATG Personalization and Scenario Development*, but this example shows how to use the RQL servlet beans.)

## RQLQueryRange: Example

```

<dsp:droplet name="/atg/dynamo/droplet/ForEach">
    <dsp:param bean="Profile.prefGenres" name="array"/>
    <dsp:oparam name="output">
        <dsp:setvalue param="prefgenre" paramvalue="element"/>
    1
    <h3><dsp:valueof param="prefgenre"/> songs: </h3><ul>
        <2>
        <dsp:droplet name="/atg/dynamo/droplet/RQLQueryRange">
            <3>
                <dsp:param name="itemDescriptor" value="song"/>
                <dsp:param name="repository"
                    value="/dynamusic/SongsRepository"/>
                <dsp:param name="howMany" value="4"/>
            3
            <4>
                <dsp:param name="queryRQL" value="genre = :prefgenre"/>
                <4>
                    <dsp:oparam name="output">
                        <li><dsp:valueof param="element.title"/> (by
                            <dsp:valueof param="element.artist.name"/>
                        </dsp:oparam>
                    </dsp:droplet>
                </ul>
            </dsp:oparam>
        </dsp:droplet>
    </dsp:oparam>
</dsp:droplet>

```

genresongs.jsp



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

1. The example code in the slide starts with a `ForEach`, to loop through the user's list of preferred genres. Recall that the contents of the `output oparam` are rendered once for each genre in the list, with the `element` output parameter being set, one by one, to each genre in `prefGenres`.
2. Within the `output oparam` of the `ForEach` loop, a new parameter is created called `prefgenre`, to be used shortly in a parameterized RQL query. Technically, "element" could have been used here rather than assigning the value of `element` to this second parameter, but that would have made the code difficult to read, because both `ForEach` and `RQLQueryRange` have an `element` output parameter.
3. Setting the `itemDescriptor` parameter to "song" means that the RQL query will return `song` items that match the query criteria. The `howMany` parameter controls how many matching items are displayed. Because the query could return hundreds of songs, this code limits how many are displayed. (If Dynamusic had wanted *all* matching songs to be displayed, `RQLQueryForEach` would be used in place of `RQLQueryRange`.)
4. The query used for the example is "`genre = :prefgenre`". In this example, Raoul has two preferences: jazz and pop. Therefore, `ForEach` will loop twice, and each time it runs `RQLQueryRange` with a different value for `prefgenre`. The first time "`genre = jazz`", the second time "`genre = pop`". The query is: return all song items whose genre property is "jazz" (or "pop" the second time through the loop).

(This code can be found in `<coursedir>/examples/chapter04/genresongs.jsp`.)

## RQL Query and Item Lookup Together

**Songs by Genre**

jazz songs:

- [Black Stockings \(by Buddy Longfellow Johnson\)](#)
- [Magic Mouse \(by Buddy Longfellow Johnson\)](#)
- [Sweetie Pie \(by Buddy Longfellow Johnson\)](#)
- [Didn't You Know? \(by Buddy Longfellow Johnson\)](#)

pop songs:

- [When I'm Young \(by The Brighton Bramblers\)](#)
- [For the Benefit of a Song \(by The Brighton Bramblers\)](#)
- [Lovely Loretta \(by The Brighton Bramblers\)](#)
- [Getting Warmer \(by The Brighton Bramblers\)](#)

**Sweetie Pie**

Song: [Sweetie Pie](#) [Download](#)

Artist: [Buddy Longfellow Johnson](#)

Album(s): [Something Wild](#)

```
... <dsp:oparam name="output">
    <li><dsp:a href="song.jsp">
        <dsp:param name="itemId" param="element.id"/>
        <dsp:valueof param="element.title"/> (by
            <dsp:valueof param="element.artist.name"/>
        </dsp:a>
    </dsp:oparam> ...
```

ORACLE

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

The example in the previous slide showed a list of songs, but the songs did not link to anything, so they were not actually very helpful for browsing. The example in this slide combines the RQL servlet bean in the previous exercise with an ItemLookup servlet bean to move from page to page.

## RQL Query and Item Lookup Together

**song.jsp**

```
<dsp:droplet name="/dynamusic/SongLookupDroplet">
  <dsp:param name="id" param="itemId"/>
  <dsp:oparam name="output">
    ...
    Song: <dsp:valueof param="element.title"/>
    ...
  </dsp:oparam>
</dsp:droplet>
```



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

The `itemId` parameter was passed in from the previous page as part of the link.

# Quiz

Which of the following query a repository to retrieve a subset of values?

- a. param
- b. RQLQueryForEach
- c. oparam
- d. RQLQueryRange



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

**Answer: d**

# Quiz

The Repository Query Language (RQL) \_\_\_\_.

- a. Uses the same syntax as SQL
- b. Is not specific to databases and can be used with any supported data source
- c. Outputs HTML
- d. Understands relationships between repository items



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

**Answer:** b, d

## Summary

In this lesson, you should have learned how to:

- Describe the Data Anywhere Architecture
- Use an ItemLookupDroplet to retrieve an item from a repository by ID
- Use RQL servlet beans to query the repository
- Use the ACC to browse and query a repository

## Additional Information

- *Repository Guide*
- *Page Developer's Guide*
  - Appendix B: ATG Servlet Beans



ORACLE

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

## Practice 4 Overview: Using Repositories

This practice covers the following topics:

- Using the ACC to browse the Songs Repository
- Creating “drill-down pages” to list artists, albums, and songs from the Songs Repository



ORACLE®

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

## Configuring SQL Repositories

ORACLE

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

# Objectives

After completing this lesson, you should be able to:

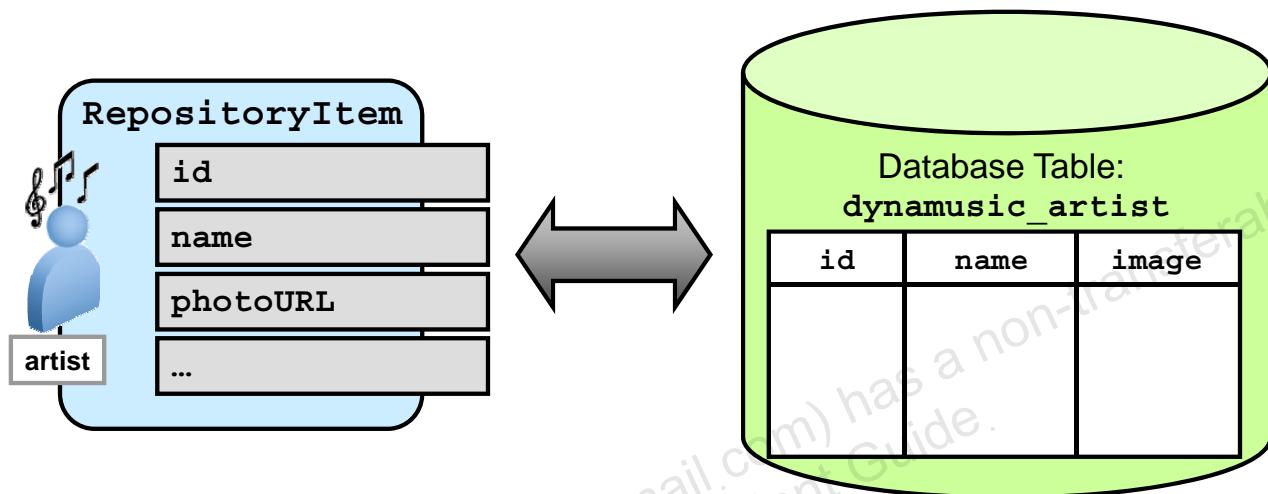
- Configure a SQL Repository component
- Create a Repository Definition file to map item properties to table columns
- Integrate a SQL Repository into the ATG Control Center (ACC)
- Use the `startSQLRepository` script to test, debug, and populate a SQL Repository



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

## Object to Relational Mapping

A SQL Repository is the Java (object) view of elements from a database.



ORACLE

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

The advantage of using the Data Anywhere Architecture to access SQL databases is that data in the database can be mapped to properties of a JavaBean, making the data easy to work with in the context of an ATG application. This lesson explains how to configure a repository that accesses data in a SQL Repository.

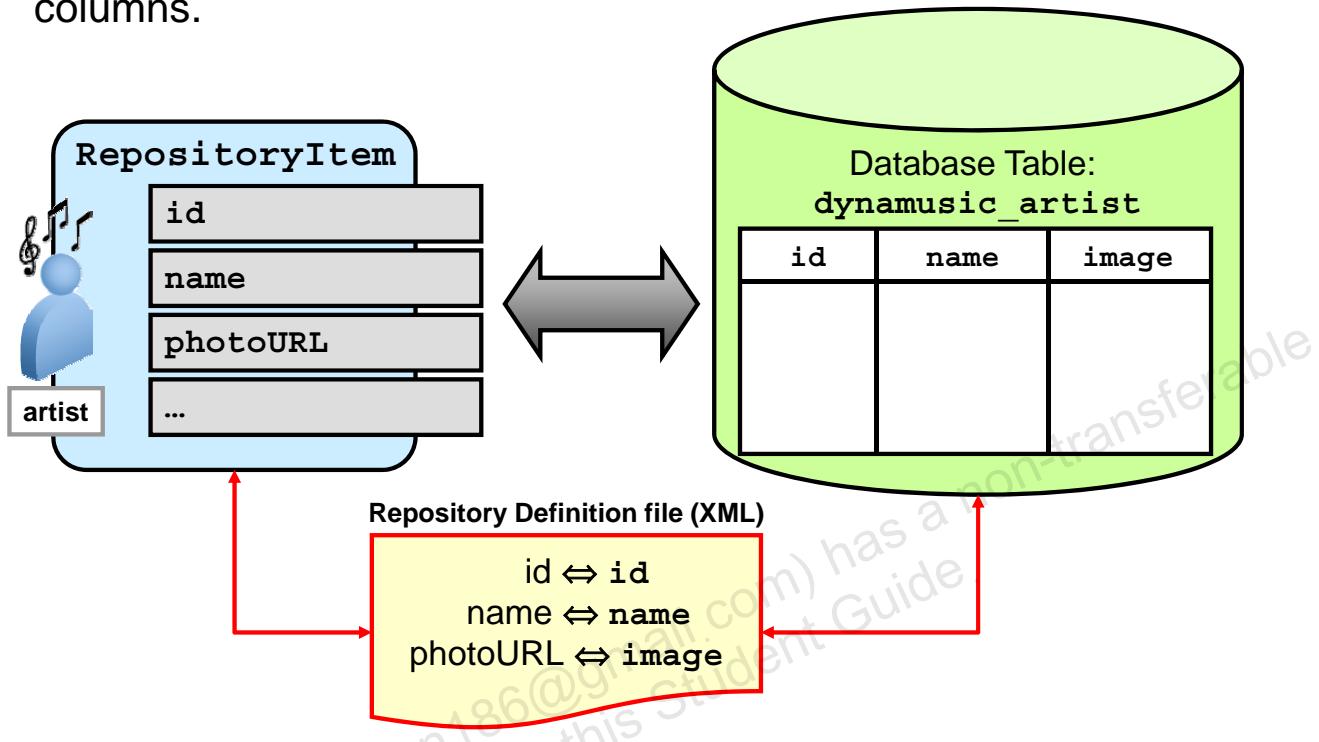
You work with `RepositoryItem` objects using the Repository API or Servlet Beans (droplets).

The actual work of accessing the Repository and converting Repository API calls into JDBC calls is the responsibility of the GSA adapter.

Note that the example in the slide shows one table for the `RepositoryItem`. In fact, a `RepositoryItem` may contain more than one database table. In SQL terms, an item is really more like a row of a join.

## Item Descriptor

An Item Descriptor maps `RepositoryItem` properties to table columns.

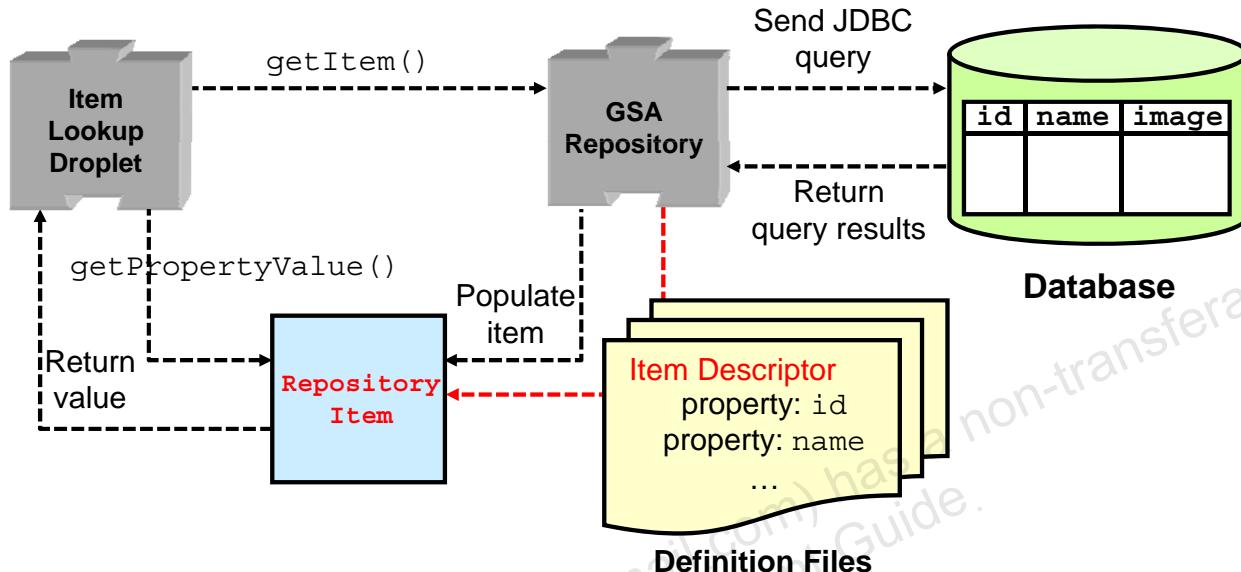


ORACLE

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

The mapping between the properties of a `RepositoryItem` and the columns of a database table is defined by an Item Descriptor. Each Item Type in the repository has a corresponding Item Descriptor in a Repository Definition file. (Often, the terms “Item Type” and “Item Descriptor” are used synonymously.)

# Repository Access to Database



ORACLE

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

1. The business logic component, an Item Lookup Droplet in the example in the slide, requests an item from the repository component: Generic SQL Adapter Connector (GSA).
2. The GSA Connector works with the Item Descriptor found in the Repository Definition file to create SQL queries against the database. (Database queries are necessary only if the item requested is not in the repository cache. SQL Repository caching is covered in the *Extending the ATG Platform* course.)
3. The GSA Connector resolves connection issues with the database and sends a query to the database based on the Item Descriptor found in the Repository Definition file.
4. Query results are returned as objects (which can be cached against future requests).
5. The business logic component now accesses the item property values it needs from the RepositoryItem.
6. The values are returned and used by the business logic component.

# Road Map

- Configuring a SQL repository
- Creating the repository definition file
- Using the `startSQLRepository` testing tool

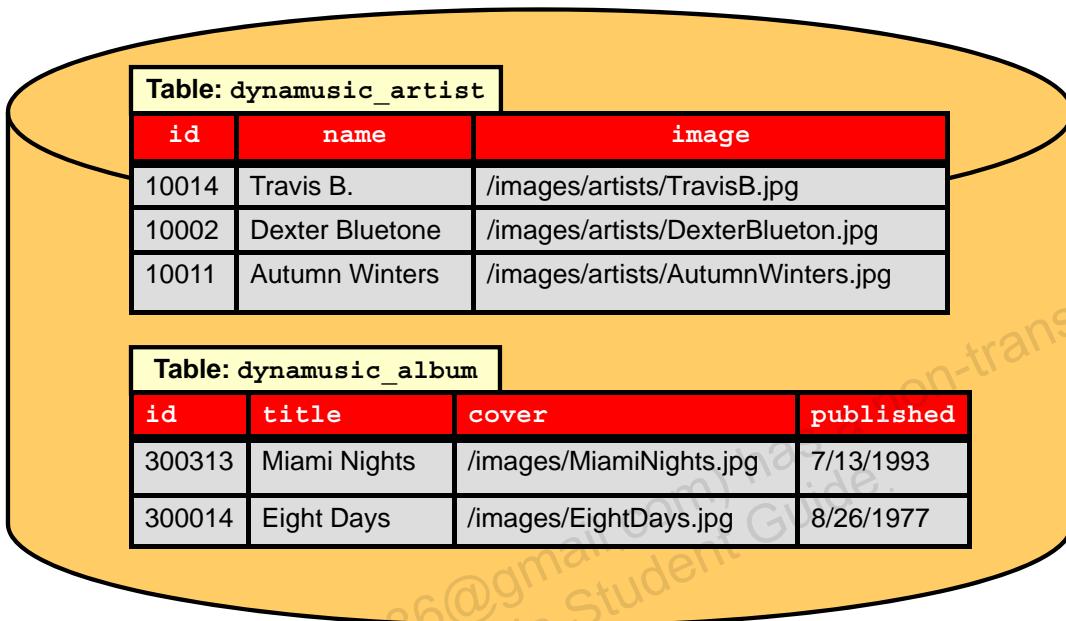


ORACLE

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

## Step 1: Set Up Database Tables

Tables must exist in the SQL database to hold the property values of repository items.



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

The ATG platform and all ATG products require certain tables to be present and populated to run. Installation of these tables is documented in the installation guide for each product and is not covered in this course.

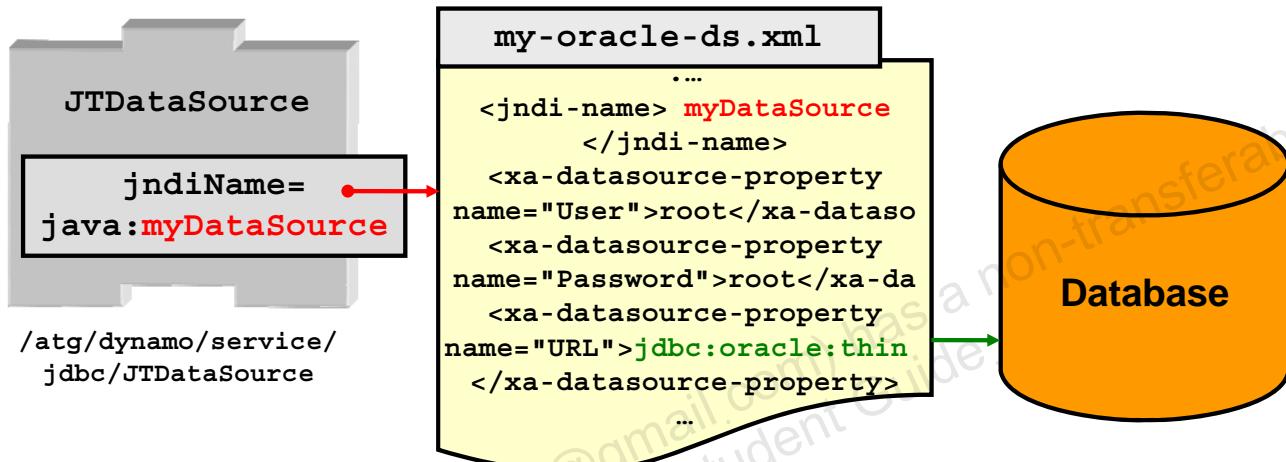
Your custom repositories will require additional tables to hold your application data. You can choose to install these tables in the same tablespace as the ATG platform tables, or in different tablespaces. In some cases, you will design the repository first, and then create database tables to support it. (ATG provides a tool to help create database tables based on repositories: `startSQLRepository -outputSQL`. For more information, refer to the Repository Guide, “Development, Testing and Debugging with the SQL Repository.”) Other times, such as when working with a legacy application, your database will already exist, and you will design your repository to access it.

In this course, all necessary database table creation and population have been done for you. The SQL scripts used to create the Songs repository can be found in the `Dynamusic-Base` module (as discussed in the practices) in the `sql` directory.

## Step 2: Configure Database Access

Your organization configures ATG to use a data source, such as Oracle.

Connection Pool (managed by the application server)



**ORACLE**

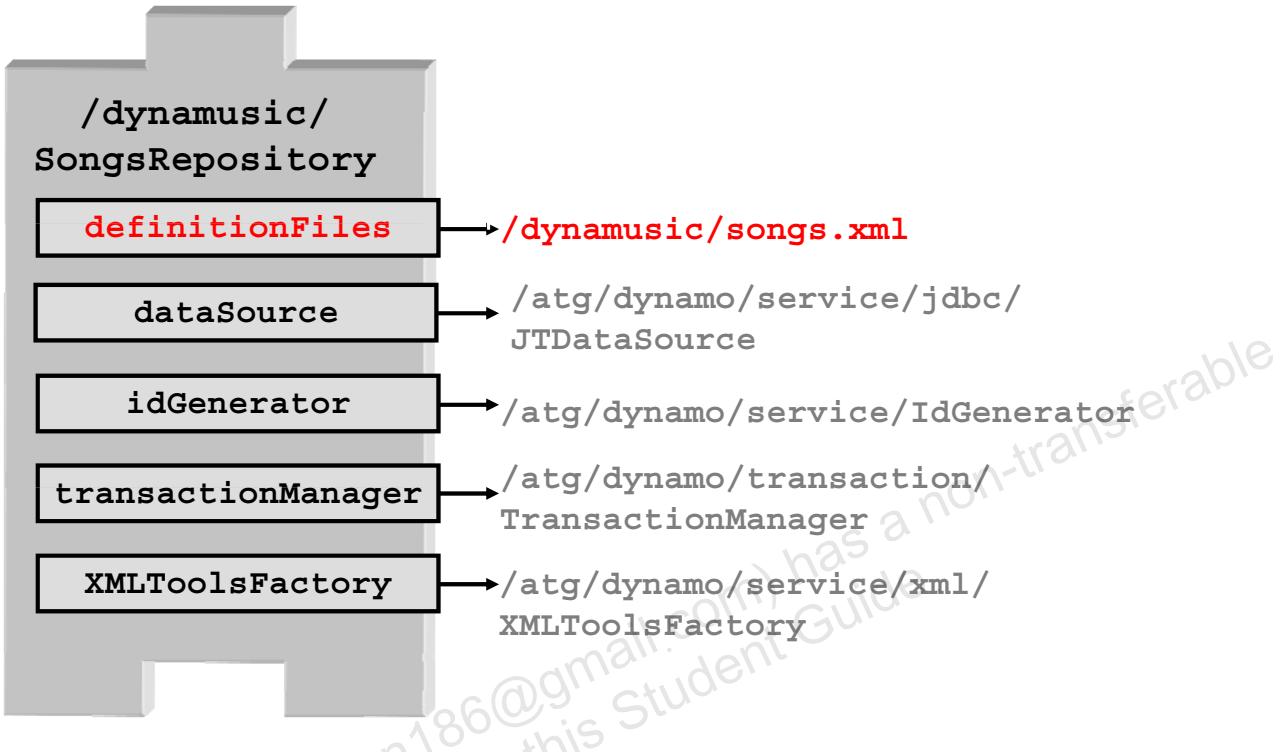
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Real world applications will need to connect to a supported database, such as Oracle. If all of your application data is in the same tablespace in the same database, you may need only one connection pool, in which case, you can use the same connection pool that the ATG platform uses, as shown in the slide. Your site may have multiple databases, or multiple tablespaces within the same database, in which case, you will need to configure a data source and connection pool for each, and then configure your repository to use the correct data source.

The default DataSource connection pool component in ATG is /atg/dynamo/service/jdbc/JTDataSource. (“JT” refers to “Java Transactions,” and the data source it points to is expected to comply with the Java Transaction API [JTA] requirements. JTA is covered in the *Extending the ATG Platform* course.)

For more information about configuring data sources, see the chapter titled “Configuring Databases and Database Access” in the *ATG Installation and Configuration Guide*, or the documentation for the application server you are using.

## Step 3: Create a SQL Repository Component



ORACLE

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

The repository component is based on the `atg.adapter.gsa.GSARespository` class.

The most important property of this object is `definitionFiles`, which is the location of the XML Repository Definition files, specified relative to the CONFIGPATH.

The following properties must also be set, but most applications use the following values:

- `dataSource`: Is a factory for Connection objects. Most repositories use the default data source: `/atg/dynamo/service/jdbc/JTDataSource`.
- `XMLToolsFactory`: Is a factory for common XML tools used to parse Repository Definition files; value: `/atg/dynamo/service/xml/XMLToolsFactory`
- `transactionManager`: Allows an application server to manage transaction boundaries; value: `/atg/dynamo/transaction/TransactionManager`
- `idGenerator`: Is the facility that generates unique IDs; value: `/atg/dynamo/service/IdGenerator`

Because these properties are required, but usually the same, an alternative to creating a repository component manually and setting these properties is to copy the “sample” SQL Repository that comes with ATG: `/atg/dynamo/service/jdbc/SQLRepository`. You can do this in the ACC, or by copying the `SQLRepository.properties` file to your project’s configuration directory.

The sample properties file for the Songs repository can be found in the course files folder under `<coursedir>/examples/chapter05/SongsRepository.properties`.

# Quiz

Names for properties in a RepositoryItem must exactly match the names in a column of a database.

- a. True
- b. False



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

**Answer: b**

## Road Map

- Configuring a SQL repository
- Creating the repository definition file
- Using the `startSQLRepository` testing tool

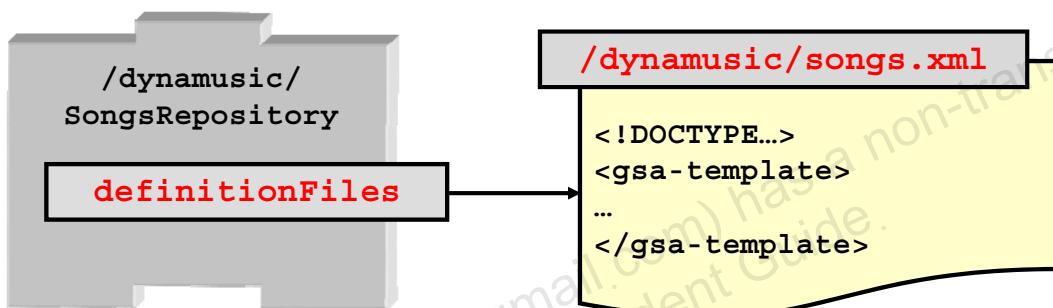


ORACLE

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

# Repository Definition File

- Is in XML format
- Describes repository items
- Maps table columns to item properties
- Is referenced by the GSARespository component, which represents the SQL Repository



ORACLE

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

The rest of this lesson uses the Dynamusic SongsRepository as an example. The full Repository Definition file for this repository is found in  
`<coursedir>/examples/chapter05/songs.xml` or in the Dynamusic-Base module.

## XML Syntax Review

- XML tags that enclose content have closing tags:  

```
<tag attr="foo"> Some content </tag>
```
- Tags that do not enclose content are closed with a “/” character:  

```
<tag attr1="foo" attr2="bar"/>
```
- Attribute values must be in double quotation marks.
- Tags can contain “child tags”:

```
<tag1 attr="foo">  
  <tag2 attr="bar"/>  
</tag1>
```



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

There are a number of good sources for more information about XML, including  
<http://www.w3.org/XML/>.

## Structure of <gsa-template>

```
<?xml version...>
<!DOCTYPE gsa-template PUBLIC "-//Art Technology
Group, Inc.//DTD Dynamo Security//EN"
"http://www.atg.com/dtds/gsa/gsa_1.0.dtd">
<gsa-template>
  <header> ... </header>
  <item-descriptor ... >
    <table ... >
      <property ... />
    </table> ...
    <property ... />
  </item-descriptor> ...
  operational tags
</gsa-template>
```



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

The example in the slide shows the structure of the repository definition file, including the `gsa-template` and `item-descriptor` tags. The operational tags are discussed later in the lesson.

Sample header:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE gsa-template PUBLIC "-//Art Technology
Group, Inc.//DTD Dynamo Security//EN"
"http://www.atg.com/dtds/gsa/gsa_1.0.dtd">
<gsa-template>
  <header>
    <name>Songs Repository</name>
    <author>Carlos Santana</author>
    <description> This defines the songs repository used by
Dynamusic.com </description>
  </header>
```

The DTD for a Repository Definition file is found in:

<atg-dir>/DAS/lib/classes.jar

The DTD is referenced using this URL:

[http://www.atg.com/dtds/gsa/gsa\\_1.0.dtd](http://www.atg.com/dtds/gsa/gsa_1.0.dtd)

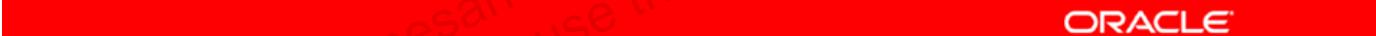
(The tag definition is always the same.)

The XML file parser automatically turns this reference into local reference of the form `dynamicsystemresource:/atg/dtds/gsa/gsa_1.0.dtd`, which can then be found in your CONFIGPATH.

If you have multiple definition files being combined, only the first file in the CONFIGPATH needs to specify this tag. The XML file combination is discussed in more detail in the lesson titled “Extending the Profile Repository.”

## Design Objects, Not Tables

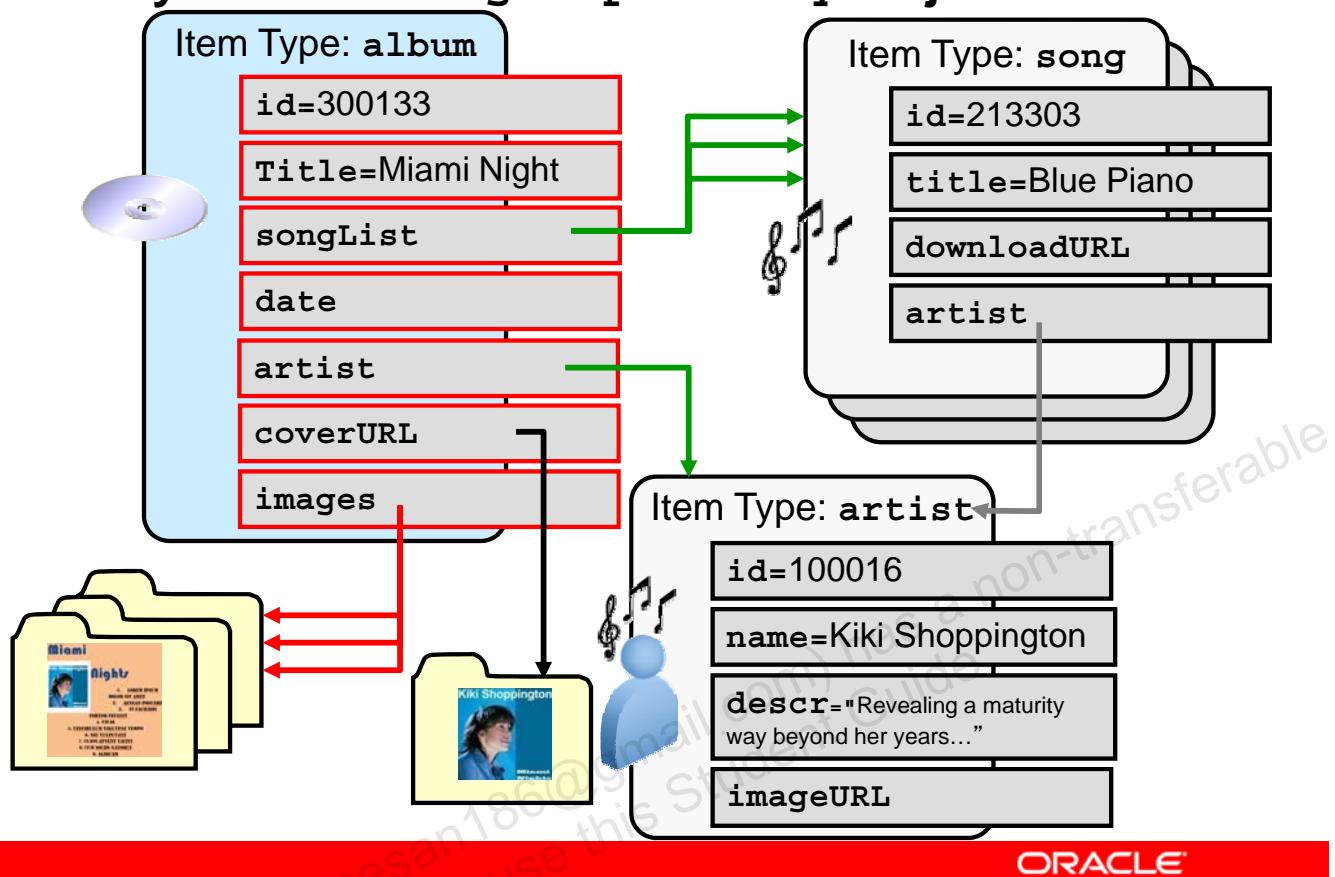
- When designing repositories for your application, focus on *object* design, rather than database structure.
  - Tables can define conglomerate data types, but do not address the data types' behavior.
  - Objects reflect data and behavior.
- Repositories persist *entities*. Design should be driven by the object model, not the table structure.



ORACLE

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

## Dynamusic SongsRepository Object Model



ORACLE

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

Before starting to build your repository, decide what business objects your application needs and their relationships. This is important to do *before* starting the implementation of the repository, as the requirements determine the details of the repository definition.

Recall the diagram from the previous lesson's slide titled "Item Relationships," showing the structure of the Songs repository.

As a source of examples, this lesson focuses primarily on the `album` repository Item Type. The important point to note is that the `album` Item Type has some simple properties (`id`, `title`, `date`, `coverURL`) and some properties that refer to other items (`artist`, `songList`).

## Item Descriptors

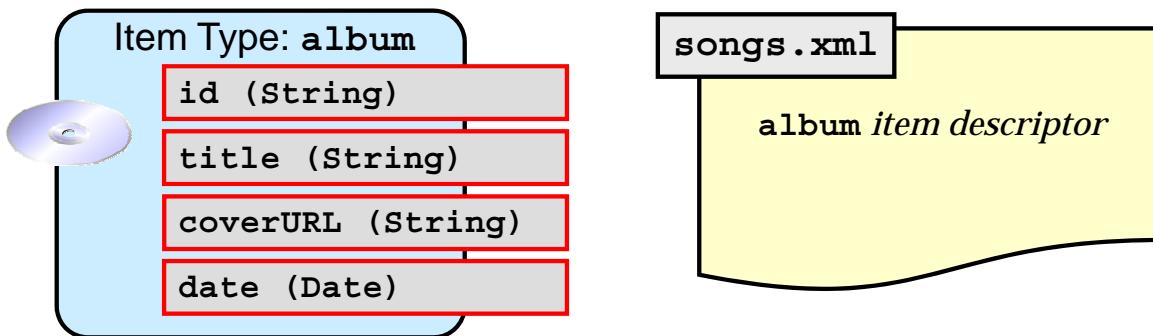


Table: dynamusic\_album

id	title	cover	published
300313	Miami Nights	/images/MiamiNights.jpg	7/13/1993
300014	Eight Days	/images/EightDays.jpg	8/26/1977
...	...	...	...

ORACLE

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

Each repository Item Type is described by one Item Descriptor, defined with the `<item-descriptor>` tag, which describes the structure of a repository item.

The discussion in the lesson starts with the “simple” properties of the album Item Descriptor: those properties that contain simple values rather than referring to other items or database tables.

## <item-descriptor> Tag

```
<?xml version...>
<!DOCTYPE gsa-template PUBLIC "-//Art Technology
Group, Inc./DTD Dynamo Security//EN"
"http://www.atg.com/dtds/gsa/gsa_1.0.dtd">
<gsa-template>
...
<item-descriptor name="album">
...
</item-descriptor>
...
</gsa-template>
```



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

## <table> Tag

```

<gsa-template>
...
<item-descriptor name="album">
  <table name="dynamusic_album" type="primary"
    id-column-name="id">
    ...
  </table>
</item-descriptor>
...
</gsa-template>

```

**Table: dynamusic\_album**

id	title	cover	published
300014	Eight Days	/images/EightDays.jpg	8/26/1977

**ORACLE**

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

The table tag has the following required attributes:

- **name:** The name of the database table
- **id-column-name:** The primary key column of the table. (This course covers only the use of tables with single primary keys. Information about using composite primary keys can be found in the “SQL Repository Data Models” chapter of the Repository Guide.) This value can be an integer (int or long) or a string (varchar).
- **type:** primary, auxiliary, or multi. Each Item Descriptor must have exactly one primary table, whose ID column will be used as the ID for the item. Additional data for the item can be stored in other tables. Multi tables will be discussed later in the lesson.

The SQL code that generated the table shown is (found in Dynamusic-Base/sql/dynamusic.sql):

```

CREATE TABLE dynamusic_album (
  id      VARCHAR(32)      not null,
  title   VARCHAR(100)       null,
  cover   VARCHAR(100)       null,
  published  TIMESTAMP        null,
  primary key(id)
);

```

## <property> Tag

```

<item-descriptor name="album">
  <table name="dynamusic_album" type="primary" id-
    column-name="id">
    <property name="id" column-name="id" data-
      type="string"/>
    <property name="title" column-name="title"
      data-type="string"/>
    <property name="coverURL" column-name="cover"
      data-type="string"/>
    <property name="date" column-name="published"
      data-type="timestamp"/>
  </table>
</item-descriptor>

```

Table: dynamusic\_album

id	title	cover	published
300014	Eight Days	/images/EightDays.jpg	8/26/1977



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

Each property tag corresponds to a property of the repository item. The major attributes are:

- name: The name used to retrieve this value from the repository item (for example, `<dsp:valueof param="element.propertyname" />`)
- data-type: The data type for this property. (The mapping between data-type values, Java object types, and SQL table column types are described in detail in the “Data Type Correspondences” section of “SQL Repository Definition Tag Reference” in the “SQL Repository Reference” chapter of the Repository Guide). Optional: The default value is “string.”
- column-name: Maps the property to a column in the table. This is optional. By default, the column-name matches the property name. Both are shown here for clarity.

Note that the table tag's id-column-name refers to the database column that contains the repository ID. You can also define a separate property for that ID, as shown in the slide example in the property tag. If you do explicitly define the ID property in the Item Descriptor, using a <property> tag, you can query repository items by their IDs and you can set a different data-type for the ID property (default is String).

## Other Types of Properties

- The properties in the previous example were single-valued, simple properties.
- Properties can be:
  - Single-valued
  - Multivalued (set, array, and so on)
- Property values can be:
  - Simple (String, integer, and so on)
  - Enumerated
  - Items (refer to other repository items)



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

All the examples shown so far have been simple, single-valued properties. This section discusses other options.

## Enumerated Properties

```

<item-descriptor name="song" display-property="title">
  <table name="dynamusic_song" type="primary" id-column-
    name="id">
    <property name="id" column-name="id" data-type="string"/>
    <property name="genre" column-name="genre" data-
      type="enumerated">
      <option value="jazz" code="10"/>
      <option value="country" code="11"/>
      <option value="pop" code="12"/>
      <option value="blues" code="13"/>
      <option value="classical" code="14"/>
    </property>
    ...
  </table>
</item-descriptor>

```

Table: dynamusic\_song

<b>id</b>	<b>title</b>	<b>genre</b>	...
13001	I'll Carry On	11	...
13210	If You Know Me	12	...
13422	Please Send Me Something	12	...

ORACLE

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

The important thing to note about enumerated properties is that the value is actually stored in the database as an *integer* corresponding to the option value. In the example in the slide, explicit numbering is used, but it is also possible to let ATG supply the numbering, in which case, the numbering starts with 0, 1, 2, and so on. The problem with following the latter method is that if you later change your repository definition for this property, you must be careful to add new enumerations at the end of the list, so as not to change the value of existing enumerations.

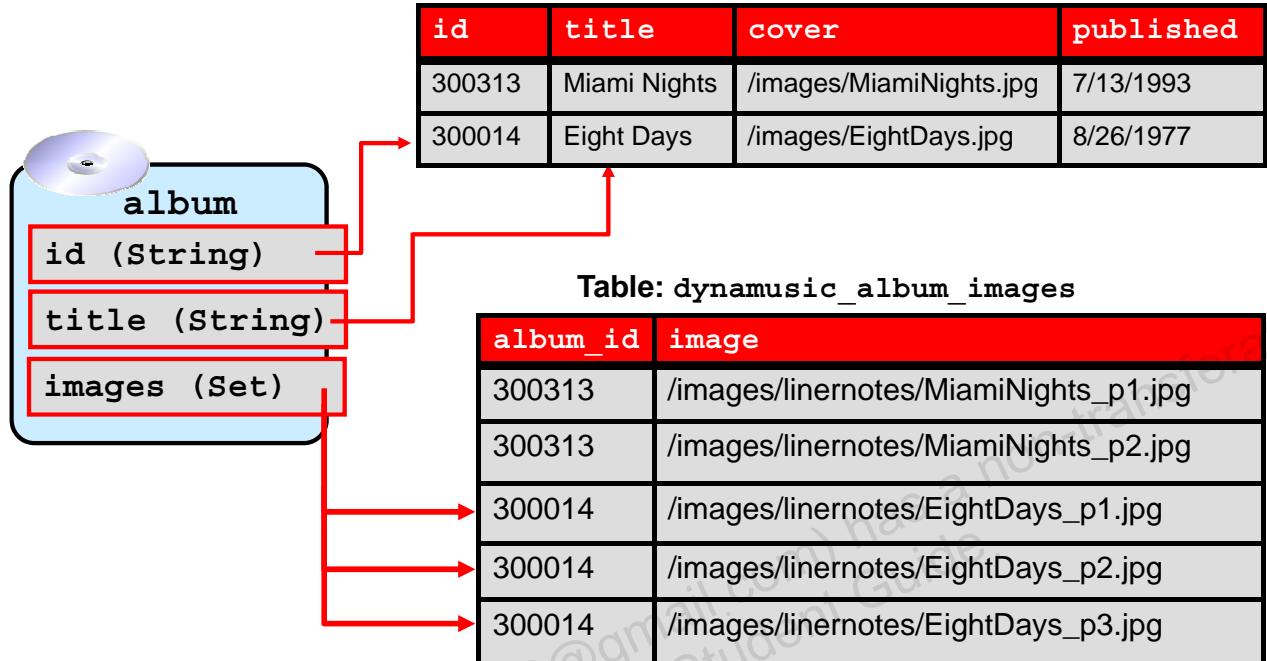
By default, when the property value is retrieved, it returns an integer. If the associated string value is needed instead (such as "jazz" or "classical"), you can add the `useCodeForValue` attribute tag:

```

<property name="genre" column-name="genre"
  data-type="enumerated">
  <attribute name="useCodeForValue" value="false"/>
  ...
</property>

```

## Multivalued Properties



ORACLE

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

Imagine that instead of just one image (the cover), Dynamusic wanted to store multiple images associated with each album, such as images from the CD inserts. A single property could hold this data if that property were of an array type or a collection (such as a List, Set, or Map).

A second table is required to hold this data. This is referred to as a *multi table*. The row in the primary table is associated with zero or more rows in the multi table through the primary key of the primary table.

```
CREATE TABLE dynamusic_album_images (
    album_id  VARCHAR(32)      not null references
dynamusic_album(id),
    image VARCHAR(100)       not null,
    primary key(album_id, image)
);
```

## Multivalued Properties: Definition File

```

<item-descriptor name="album">
  <table name="dynamusic_album" type="primary" id-
    column-name="id">
    <property name="id" column-name="id" data-
      type="string"/>
    ...
  </table>
  <table name="dynamusic_album_images" type="multi"
    id-column-name="album_id">
    <property name="images" data-type="set"
      component-data-type="string" column-
      name="image"/>
  </table>
</item-descriptor>

```



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

For single-value properties, you need to set data-type to a simple value, such as a string or an integer. For multivalue properties, data-type is set to a collection type or array (such as array, set, or map). To indicate the type of simple values within the collection, use component-data-type.

The “id” column of a multi table is generally a reference to the primary key of the primary table. In this example, the primary table is dynamusic\_album with a primary key column ID, and the multi table is dynamusic\_album\_images with an “id” column that is a foreign key reference to dynamusic\_album:id.

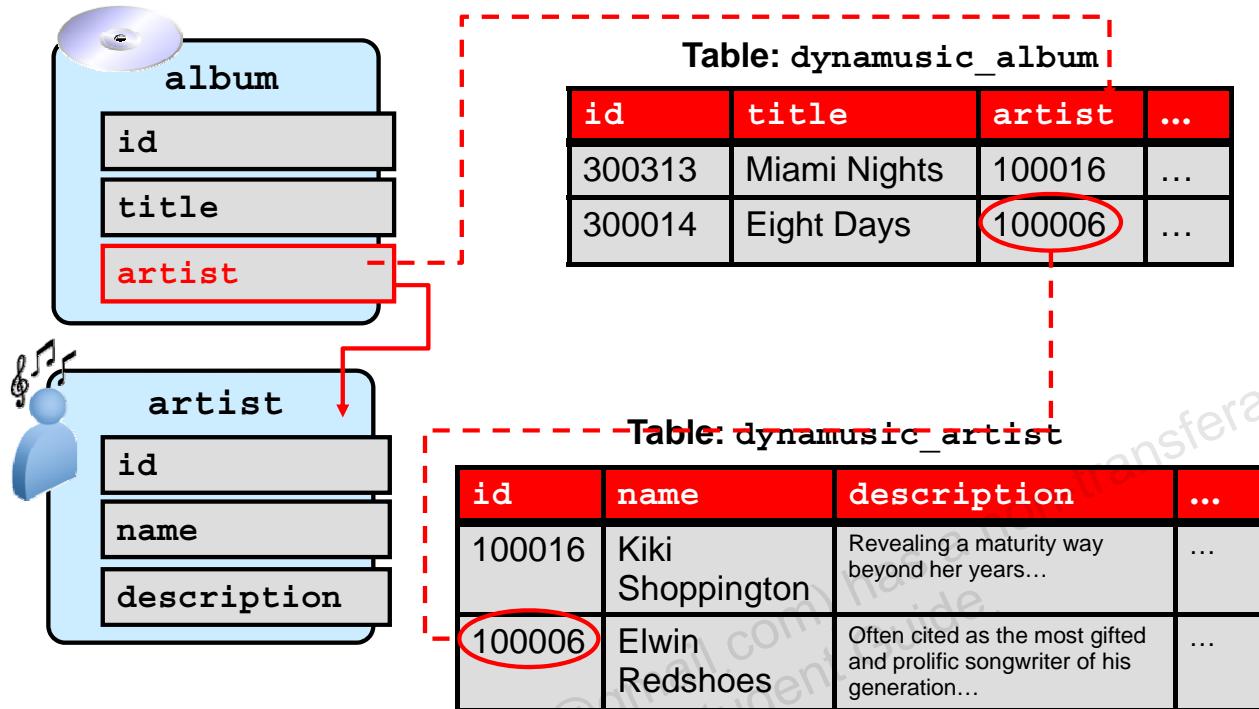
```

CREATE TABLE dynamusic_album (
  id      VARCHAR(32)      not null,
  ...
  primary key(id)
);

CREATE TABLE dynamusic_album_images (
  album_id      VARCHAR(32)      not null references
  dynamusic_album(id),
  ...
  primary key(album_id, image)
);

```

## Item Properties



ORACLE

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

The example in the slide focuses on the `artist` property of the `album` Item Type. This is a single property, which points to another repository item (of type `artist`). The database tables are set up so that the column in the `album` table associated with the `artist` property (in this case, the `artist` column) contains the ID of the row in the `artist` table associated with this album. Often this is defined as a foreign key reference in the table definition, as shown in the following:

```

CREATE TABLE dynamusic_album (
    id      VARCHAR(32) not null,
    title   VARCHAR(100)      null,
    artist     VARCHAR(32) null references dynamusic_artist(id),
    ...
    primary key(id)
);

CREATE TABLE dynamusic_artist (
    id      VARCHAR(32) not null,
    name   VARCHAR(100)      null,
    description LONG VARCHAR      null,
    image    VARCHAR(32) null,
    primary key(id)
);
  
```

## Item Properties: Definition File

```
<item-descriptor name="album">
  <table name="dynamusic_album" type="primary" id-
    column-name="id">
    <property name="id" column-name="id" data-
      type="string"/>
    <property name="title" column-name="title"
      data-type="string"/>
    <property name="artist" column-name="artist"
      item-type="artist"/>
    ...
  </table>
</item-descriptor>
```

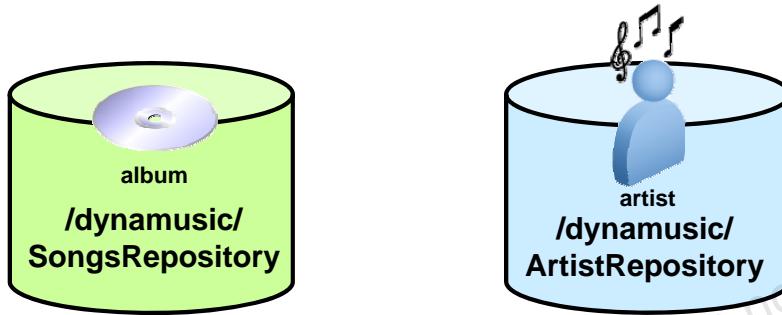


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The important thing to note in the code example in the slide is that instead of having a data-type defined for this property, an item-type is defined. This marks the property as being a reference to another repository item, rather than a “simple” property containing a Java primitive or object.

## Cross-repository Item Properties

- An item-type property can refer to an Item Type defined in a separate repository.
- What if artists and albums were in separate repositories?



```
<property name="artist" column-name="artist"
          item-type="artist"
          repository="/dynamusic/ArtistRepository" />
```

ORACLE

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

Recall that in ATG, a repository is *not* a database itself, but rather an object mapping into a database (or other type of data storage). The figures in the slide superimpose the Nucleus name of a GSARespository Nucleus component on top of the database as a compact way to represent all the parts functioning together.

## Multi-item Properties: One-to-Many

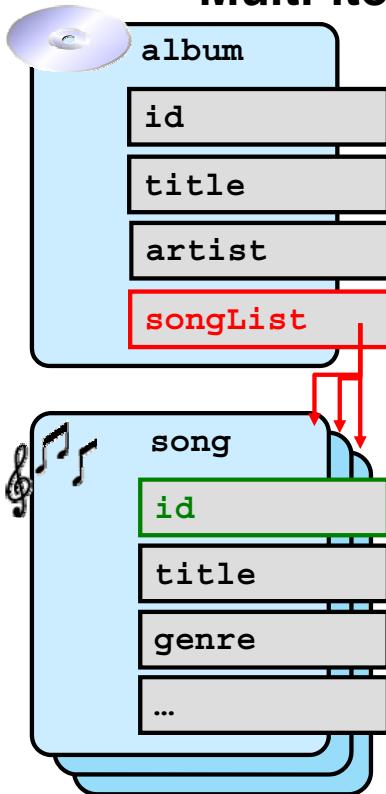


Table: dynamusic_album			
id	title	artist	...
300313	Miami Nights	100016	
300014	Eight Days	100006	...

Table: dynamusic_album_songs	
album_id	song_id
300014	204101
300014	204103
310023	13001

Table: dynamusic_song				
id	title	...	genre	...
208102	From This Moment On		11	...
204101	Tess		12	...
204103	Watching the Waves		12	...

ORACLE

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

The example in the slide combines the previous two ideas: a multiproperty where each of the elements is a repository item reference rather than a simple property value. An album has a set of associated songs. As with the previous example, a `multi` table is used, but this time, the second column in the table is not the property value itself, but rather, an ID reference to the table containing the songs. Sometimes this table is referred to as a “glue” table because it “glues” the `album` table to the `song` table.

```

CREATE TABLE dynamusic_album (
    id      VARCHAR(32)      not null,
    title  VARCHAR(100)       null,
    primary key(id)
);

CREATE TABLE dynamusic_album_songs (
    album_id  VARCHAR(32)      not null references
dynamusic_album(id),
    song_id   VARCHAR(32)      not null references
dynamusic_song(id),
    primary key(album_id, song_list)
);

```

```
CREATE TABLE dynamusic_song (
    id      VARCHAR(32)      not null,
    title  VARCHAR(100)       null,
    genre   INTEGER          null,
    ...
);
```

It is also possible to handle this relationship with just two tables, by having an `album` column in the `song` table. (This would make a many-to-many relationship between albums and songs impossible though.) If it were set up that way, the code in the next slide would simply refer to `dynamusic_song` instead of `dynamusic_album_song`.

## Multi-item Properties: One-to-Many in the Definition File

```
<item-descriptor name="album">
  <table name="dynamusic_album" type="primary" id-column-
    name="id">
    <property name="id" column-name="id" data-
      type="string"/>
    ...
  </table>
  <table name="dynamusic_album_songs" type="multi" id-
    column-name="album_id">
    <property name="songList" data-type="set"
      component-item-type="song"
      column-name="song_id"/>
  </table>
</item-descriptor>
```



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

Note the `component-item-type` attribute of the `property` tag in the example in the slide. In the previous example of multivalue properties, in which the values were simple strings, the `component-data-type` attribute was used instead.

In this example, the `data-type` is `set` (which maps to a Java type of `java.util.Set`). Other types, such as a `Map` or an array, are also supported, but the syntax and table setup are slightly different. This example also shows only a one-to-many relationship. Many-to-many relationships are also supported. For details about the syntax of these other types of relationship definitions, including defining many-to-many relationships, attend the follow-on to this course, *Extending the ATG Platform*. Also see the *ATG Repository Guide* chapter titled “SQL Repository Data Models.”

In the previous slide, it was mentioned that two tables could be used instead of three by adding an `album_id` column to `dynamusic_song`. If the database had been arranged that way, this code would be identical except that where the `dynamusic_album_songs` table is specified, `dynamusic_song` would be specified instead.

## Types of Multivalue Properties

- The previous example used a Set for the multivalued property type.
  - A Set holds unordered data.
- Other multivalue types are supported.
  - List: Ordered collection
  - Map: Collection with key/value pairs
  - Array: Ordered, numbered data
- What types of data are suited to these different multivalue data types?

ORACLE

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

The decision about what type to use depends on your application's requirements. Set is the easiest data type to use, which is why it is used as an example in the lesson. Your application may require data in a different form. In the examples shown in, images associated with an album were stored in an unordered set. What if the images were always from the CD insert liner notes, and Dynamusic wanted to be able to display them by page number ("click on the number to see that page of the CD inserts...")? What data type would support that functionality? Alternatively, what if Dynamusic had different types of images available and wanted to be able to display them by type, such as "front cover," "back cover," "promotion poster," and so on? What data type would support labeling each image according to type?

Types of multiproperties other than "set" require a third column in the "glue" table to hold the key (in the case of a Map) or the index (in the case of an array) or the item order (in the case of a List.) That third column is specified in the <table> tag by using the multi-column-name attribute.

# Quiz

How many tables can be contained in an Item Descriptor?

- a. 1
- b. 2
- c. 5
- d. As many as you want



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

**Answer: d**

# Quiz

An item-type property can refer only to an Item Type defined in the current repository.

- a. True
- b. False

 ORACLE

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

**Answer: b**

## Item Descriptor Inheritance

Item types can be created as subtypes of other Item Types.

- Supertype properties are inherited.
- The subtype may add properties stored in an auxiliary or a multi table.
- It is similar to object-oriented subclassing.

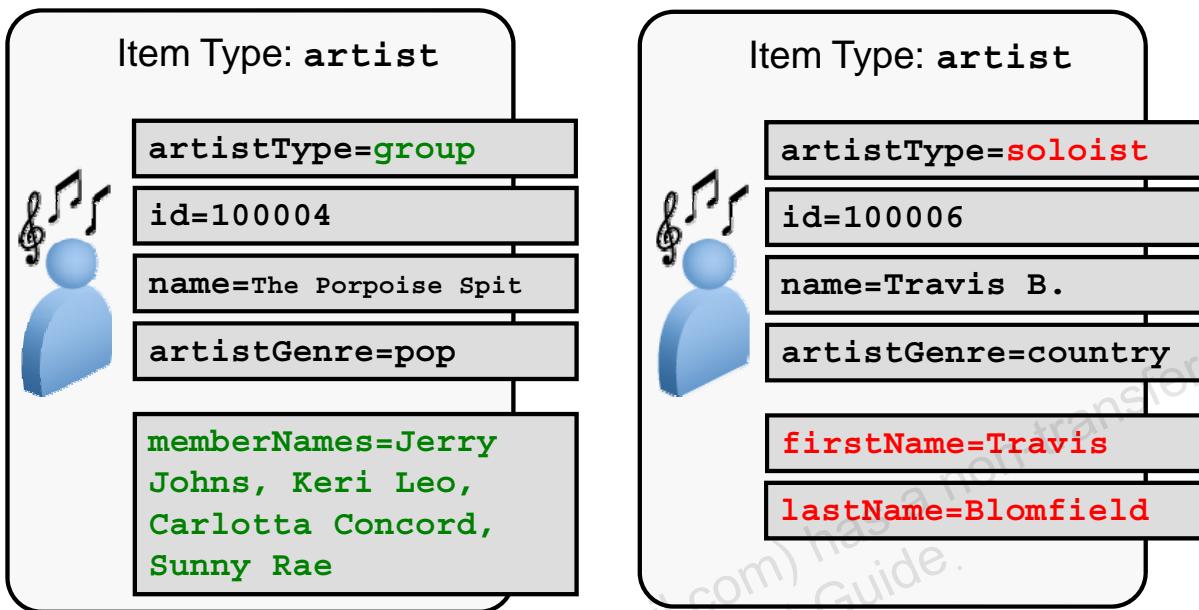


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

### Multilevel Subtyping

- Each subtype that adds additional properties to the base type requires an additional auxiliary or multi table, which will be joined with the tables associated with the base type. This means that if you have many levels deep of subtypes, you have to join many tables, which can have a marked impact on performance.
- No multiple inheritance is supported. A subtype can have only one supertype.

## Dynamusic Artist Subtypes



ORACLE

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

Dynamusic artists can either be soloists (individual musicians) or groups (such as bands) [**artistType=soloist** or **artistType=group**]. All artists have certain properties in common, such as genre and description. Some properties, however, differ. A group has both a group name and members of the group. An individual soloist could have a stage name that is different from his or her real name.

The data is stored in three tables: **dynamusic\_artist**, **dynamusic\_group**, and **dynamusic\_soloist**. The primary table is **dynamusic\_artist**; **dynamusic\_group** is a multi table and **dynamusic\_soloist** is an auxiliary table.

**Note:** In a real-world example, you would probably link the member names to the soloist table and take advantage of the first name and last name columns. For the sake of simplicity, this was not done in the example in the slide.

## Subtypes in the Database: Example

Table: dynamusic\_artist

<b>id</b>	<b>name</b>	<b>artist_genre</b>	<b>artist_type</b>
100004	The Porpoise Spit	pop	1 (group)
100006	Elwin Redshoes	pop	2 (soloist)
100012	Travis B.	country	2 (soloist)

Table: dynamusic\_group

<b>id</b>	<b>member_name</b>
100004	Jerry Johns
100004	Keri Leo
100004	Carlotta Concord
100004	Sunny Rae

Table: dynamusic\_soloist

<b>id</b>	<b>first_name</b>	<b>last_name</b>
100006	Elwin	Redshoes
100012	Travis	Blomfield

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

The dynamusic\_artist table is a *primary* table—that is, every artist (whether group or soloist) must have exactly one row in the dynamusic table. dynamusic\_soloist is an auxiliary table—artists may or may not have a row in the table. dynamusic\_group is a multi table because a group has multiple members.

The artist\_type column is an enumerated property, so an integer would appear in the database column. The example has three enumerated values:

- 0=artist
- 1=group
- 2=soloist

In the example in the slide, the string is placed in parentheses in the column for clarity only, and is not stored in the database column with the integer.

## Base Item Descriptor: artist

1. Add the sub-type-property attribute to Item Descriptor.
2. Add enumerated property for types.

```

<item-descriptor name="artist"
    sub-type-property="artistType">1
    <table name="dynamusic_artist" type="primary"
        id-column-name="id" >
        <property name="name" column-name="name"
            data-type="string"/>
        <property name="artistGenre" column-name="genre"
            data-type="string"/>
        <property name="artistType" data-type="enumerated"
            column-name="artist_type" writable="false">2
            <attribute name="useCodeForValue" value="false"/>
            <option value="group" code="1"/>
            <option value="soloist" code="2"/>
        </property>
    </table>
</item-descriptor>

```



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

1. sub-type-property: The name of the property that is used to indicate the type of the item (in this example, artistType).
2. type property: This property (named in sub-type-property) is an enumerated property whose possible values include all the possible subtypes of this item. You should set writable="false" and hidden="true" on the type property because the value should not be changed or viewed in the ACC. After you set the "type" of an item, it is fixed.

**Note:** In the example in the slide, you have two possible types: group and soloist. With this design, users could still create a generic “artist” item but if they do, there will not be any entry in the artistType property, and this is considered bad practice. Therefore, if your application needs to create items of the supertype, then the supertype should also be listed in an <option > tag of your sub-type-property.

The SQL code to configure the table for this example might look like this:

```

CREATE TABLE dynamusic_artist (
    id      VARCHAR(32)  not null,
    name    VARCHAR(100) null,
    artist_type INTEGER    null,
    primary key(id)
);

```

Note that the artist-type column is an integer. This is because enumerated properties are stored as integer “codes” in the database corresponding to the option value selected. By default, the codes are 0–n, although you may want to explicitly define the codes in the XML. That way, you have full control over values, especially if you add a new option value in the future.

## Subtype Item Descriptor: artistType

The Item Descriptor for a subtype item has:

- super-type attribute
- sub-type-value attribute

```
<item-descriptor name="soloist" super-type="artist"
    sub-type-value="soloist">
    <table name="dynamusic_soloist" type="auxiliary"
        id-column-name="id">
        <property name="firstName"
            column-name="first_name" data-type="string"/>
        <property name="lastName"
            column-name="last_name" data-type="string"/>
    </table>
</item-descriptor>
```



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

sub-type-value: The name of this subtype (must correspond to one of the value options in the supertype's type property)

super-type: The name of the supertype item

The SQL code that configures this table might look like the following:

```
CREATE TABLE dynamusic_soloist (
    artist_id      VARCHAR(32)      not null,
    first_name     VARCHAR(100)      null,
    last_name      VARCHAR(100)      null,
    primary key(artist_id)
);
```

The XML for the group subtype includes a one-to-many relationship. The following is the XML configuration for group:

```
<item-descriptor name="group" super-type="artist"
    sub-type-value="group">
    <table name="dynamusic_artist_group_members"
        type="multi" id-column-name="artist_id">
        <property name="memberNames" data-type="set"
            component-data-type="string"
            column-name="member_name"/>
    </table>
</item-descriptor>
```

The SQL code to configure the table might look like the following:

```
CREATE TABLE dynamusic_artist_group_members (
    artist_id  VARCHAR(32)  not null references
        dynamusic_artist(id),
    member_name VARCHAR(32)  not null,
    primary key(artist_id,member_name)
);
```

## Advantages of Subtyping

Items can:

- Share properties without duplicating configuration code
- Be queried by supertype or subtype

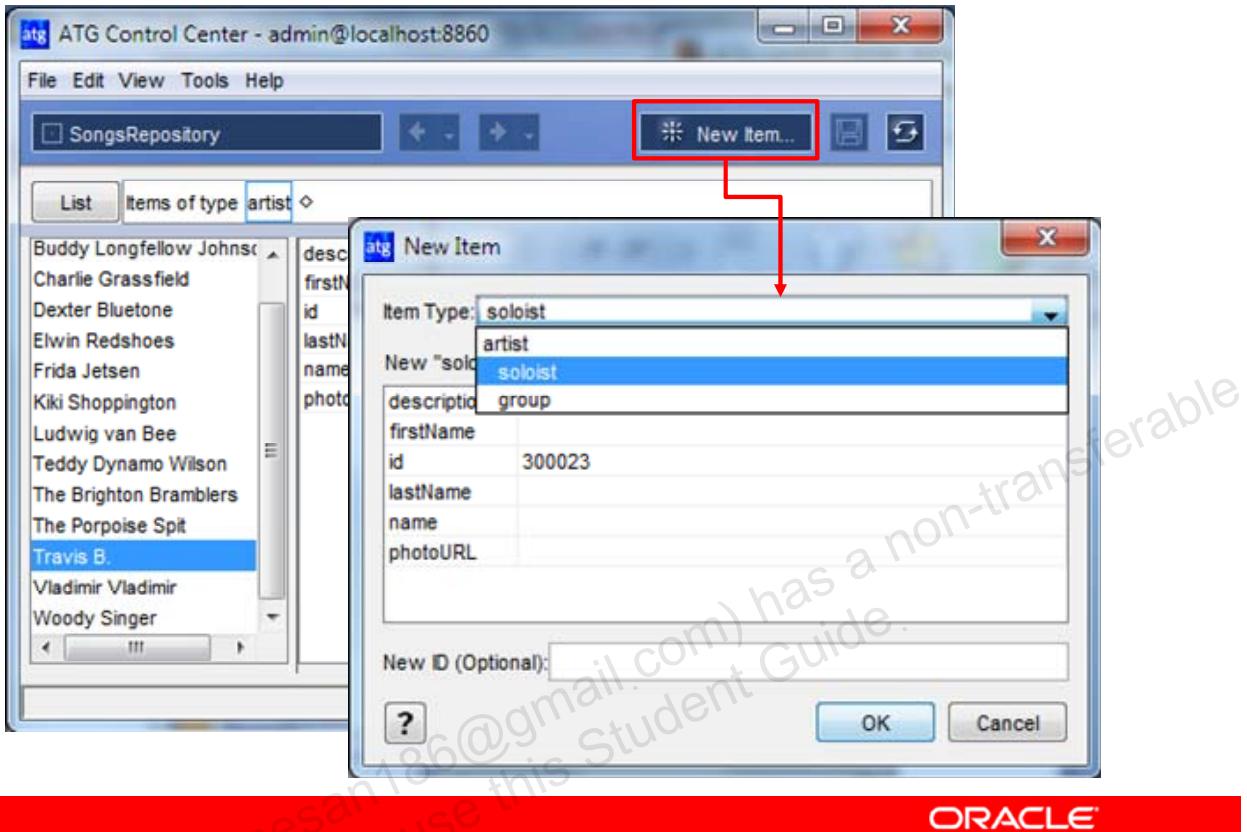
Soloist	Frida Jetsen	song	description	The almost impossibly
	Kiki Shoppington	artist	firstName	Travis
	Ludwig van Bee	group	lastName	300020
	Teddy Dynamo Wilson	soloist	albumName	Blomfield
	The Brighton Bramblers	album	photoURL	Travis B.
Group	The Porpoise Spit			/images/TravisB.jpg
	Travis B.			
	Vladimir Vladimir			

ORACLE®

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

Another advantage of subtyping involves targeters and slots. Targeters and slots are constrained to being a homogeneous list of a single type of repository item (for example, a list of albums or a list of songs). If a supertype called “recording” was created, of which albums and songs were both subtypes, then a slot or targeter could be set to be a list of recordings, and therefore, have both albums and songs on the list.

# Creating a Subtype Item



ORACLE

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

When you create a new item in the ACC, you can select the subtype you would like to create. The one you select determines which properties appear in the property editor. This also holds true in the Business Control Center (BCC) as well.

## ID Generation

- By default, each Item Type is in its own “ID space.”
  - The ID space name is the name of the Item Type.
  - The ID space can be autocreated or explicitly configured in /atg/dynamo/service/idspaces.xml.
- A specific ID space for an Item Type can be specified using the id-space-name attribute.
- Settings such as batch size, prefix, and seed number can also be specified in idspaces.xml.



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

By default, SQL Repositories use ATG's /atg/dynamo/services/idGenerator service to generate unique IDs for repository items.

The ID spaces used by the default ID Generator are configured in <CONFIGPATH>/atg/dynamo/service/idspaces.xml. If an ID space is used which is not in idspaces.xml, and the ID Generator's autocreate property is true, then an ID space with that name is automatically created using the default ID space configuration.

By default, each repository Item Type uses a different ID space, whose name is the same as the name of the Item Type. (That is, the “articles” Item Type would default to use the “articles” ID space.) If necessary, that ID space would be autocreated.

In some cases, you may want to configure a repository Item Type to use an ID space other than one equal to the name of the Item Type: if you have multiple repositories with Item Types of the same name (for example, “folder”), and you do not want them to share an ID space; if you have multiple Item Types with different names, but you want them to share the same ID space (so that no two items of those types will overlap an ID).

### Sample XML Code

```
<item-descriptor name="song" id-space-name="dynamusic_songs">  
...  
</item-descriptor>
```

For more information about ID Generation, see the Programming Guide, “Core Dynamo Services,” under “ID Generators.”

## Grouping Properties

- To improve performance, you can group properties.
  - Properties in the same group are retrieved in the same SELECT statement.
- By default, properties in the same table are in the same group.
- Specify a different group name by using the group property attribute.



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

Grouping properties can improve performance by making sure that properties that are:

- Frequently used together are retrieved at the same time
- Rarely used are *not* retrieved when more frequently used properties are

For example, if an application makes frequent use of a user's mailing address, you may want to load the address into cache whenever you request the user's name and login. By default, if the address information is in a separate table from the name information, it will not be loaded when the name property is loaded. However, you can put the name information and address information in the same group to force "pre-fetching" of the address.

Conversely, you may not want a property loaded before it is needed. For example, if an application stores a picture of each user in the primary table, but that picture is not used frequently (and it is expensive to load because it is so large) you may want to put it in a different group than the rest of the properties from that table. The following slide shows the XML for this situation.

For more information, see the *ATG Repository Guide*, "SQL Repository Item Properties," under "Property Fetching."

## Example: Grouping Properties

In the following example, if a JSP needs to display the user's name, but does *not* need the user's portrait, the portrait property will not be retrieved:

```
<item-descriptor name="user">
  <table name="dps_user" type="primary"
    id-column-name="id">
    <property name="firstName" group="name"/>
    <property name="lastName" group="name"/>
    <property name="portrait" group="picture"/>
  </table>
</item-descriptor>
```



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

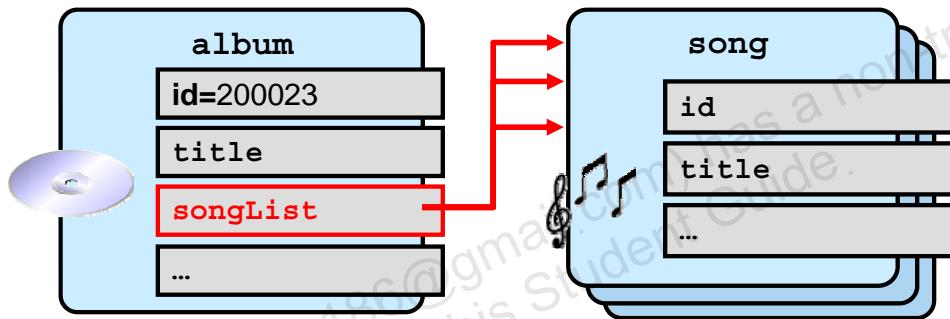
Note that the specific values assigned to the group attribute are up to you. The point is simply to delineate which properties have the same group value and which have different group values.

For more information, see the *ATG Repository Guide*, “SQL Repository Item Properties,” under “Property Fetching.”

# Cascading Relationships

When an item property refers to another item, you can delete, insert, or update both items at the same time:

```
<item-descriptor name="album" ... >
  <table name="dynamusic_album_songs" ... >
    <property name="songList" data-type="set"
      component-item-type="song"
      column-name="song_list"
      cascade="delete"/>
  </table> ...
```



ORACLE

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

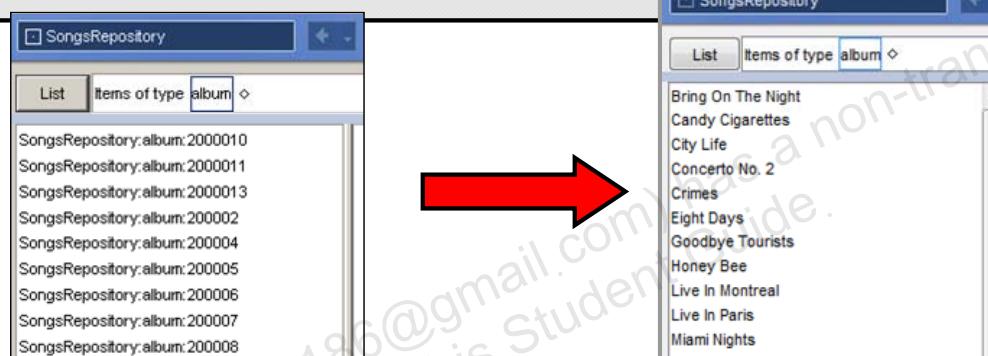
Cascading deletes, updates, and inserts allow you to modify an item and have all associated items automatically updated, inserted, or deleted at the same time.

Other options for cascade include “update” or “insert.” You can combine these settings as in `cascade="insert, delete"`. You can also control the order of deletion of the items to accommodate foreign key references in the database as necessary using `cascadeDeleteOrder`.

For more information, see the *ATG Repository Guide*, “SQL Repository Data Models,” under “Cascading Data Relationships.”

## Display Property Attribute

```
<item-descriptor name="album" display-property="title">
<table name="dynamusic_album" type="primary" id-column-name="id">
  <property name="id" column-name="id" data-type="string"/>
  <property name="title" column-name="title" data-type="string"/> ...
</table>
</item-descriptor>
```



ORACLE

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

By default, when the ACC displays a list of repository items, it lists them by repository ID. This is not very convenient. To improve the usability of your repository, it is helpful to designate a “display property” for every Item Type. That way, whenever the ACC needs to display a repository item of that type, it displays the value of that property (if the property is set) instead of the repository ID. To do that, add a `display-property` attribute to the `<item-descriptor>` tag.

**Note:** This also affects how repository items appear in the BCC.

## Operational Tags

Repository Definition files can also contain operational tags, which are useful for testing or populating the repository:

- <add-item>
- <update-item>
- <remove-item>
- <query-items>



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

Be careful with using the operational tags in production systems, because the operations is executed every time the Repository component is first started. Sometimes it can be helpful to have a separate Repository Definition file that contains just operation tags for populating, modifying, or testing a database. These can be temporarily added to the repository component's definitionFiles property, or you can use the following to run the file from the command line:

```
startSQLRepository -m yourModule file repository-definition-file.xml
```

Details about all operational tags are in the “SQL Repository Reference” chapter of the *ATG Repository Guide*. Details about the startSQLRepository script are in the “Developing and Testing an SQL Repository” chapter.

## Repository Definition Summary

- <item-descriptor> tag
  - One for each Item Type in the repository
  - Use the display-property attribute for ease of use in the ACC.
- <table> tag
  - One primary table per Item Type
  - Any number of multi tables
  - Any number of auxiliary tables



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

auxiliary and multi tables are joined with the primary table on the “id” column, which must be a foreign key reference to the primary table.

## Repository Definition Summary

- <property> tag:
  - It maps a table column to a RepositoryItem property.
  - Use the data-type attribute for simple values or item-type attribute for item relationships.
  - Use component-data-type for multivalued properties storing simple values; use component-item-type for one-to-many item relationships.



ORACLE

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

# Quiz

Advantages of using subtypes include:

- a. All subitems share all properties with each other.
- b. Items may share properties without duplicating configuration code.
- c. Items can be queried by supertype or subtype.



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

**Answer: b, c**

# Quiz

Identify the advantages of grouping properties.

- a. Properties in the same group are retrieved only when each one is used.
- b. Properties that are frequently used together are retrieved at the same time.
- c. Properties that are rarely used are *not* retrieved when more frequently used properties are.
- d. Properties that are grouped are cached.



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

**Answer:** b, c

## Road Map

- Configuring a SQL repository
- Creating the repository definition file
- Using the `startSQLRepository` testing tool

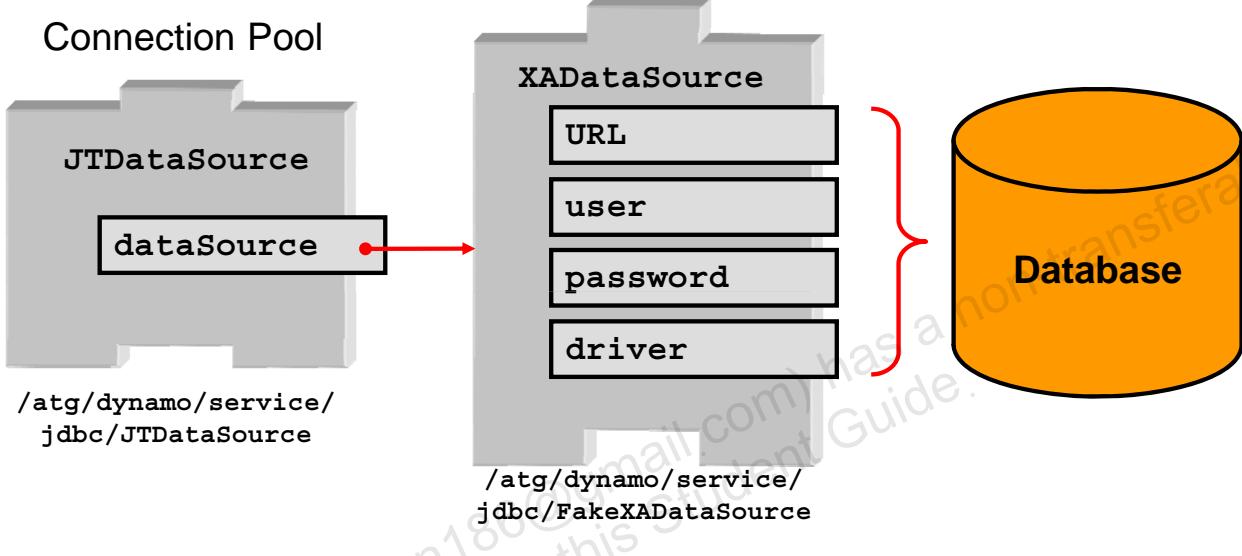


ORACLE

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

# Setting Up a Test Data Source

- ATG provides tools to import to, export from, and test repositories.
- First set up a test data source.



ORACLE®

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

When your ATG application is running on a supported application server, it uses a data source connection pool provided by the application server, as shown in previous slides. However, the `startSQLRepository` and import and export scripts run as stand-alone applications, and therefore, do not have access to application server resources such as data sources. To use these tools:

1. Create an additional configuration layer, such as in a test module.
2. In the test configuration layer, create a data source component (by convention `/atg/dynamo/service/jdbc/FakeXADatasource`) using for a base class `atg.service.jdbc.FakeXADatasource`.
3. Redefine the `/atg/dynamo/service/jdbc/JTDataSource` component (and any other connection pools your application uses) using base class `atg.service.jdbc.MonitoredDataSource` and set its `dataSource` property to the data source component created in step 2.

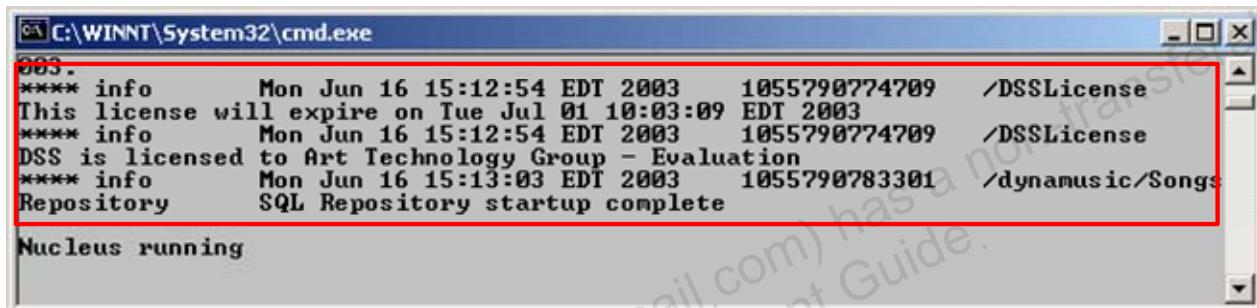
In this course, a separate module (called `TestDynamusic`) is provided for you to easily test your repository configurations in the practices.

## Testing a SQL Repository

- To test a repository without restarting ATG:

```
cd <atgdir>/home/bin  
startSQLRepository -m TestDynamusic  
-repository /dynamusic/SongsRepository
```

- Ensure that your database is started first.



```
C:\>C:\WINNT\System32\cmd.exe  
883.  
**** info Mon Jun 16 15:12:54 EDT 2003 1055790774709 /DSSLicense  
This license will expire on Tue Jul 01 10:03:09 EDT 2003  
**** info Mon Jun 16 15:12:54 EDT 2003 1055790774709 /DSSLicense  
DSS is licensed to Art Technology Group - Evaluation  
**** info Mon Jun 16 15:13:03 EDT 2003 1055790783301 /dynamusic/Songs  
Repository SQL Repository startup complete  
  
Nucleus running
```

ORACLE

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

startSQLRepository is a valuable testing and development tool. The command in the slide tests the connection to the database and validates the Repository Definition file. startSQLRepository has other options to allow you to import or export data from the repository, generate SQL scripts to create the schema associated with a repository and other useful functions.

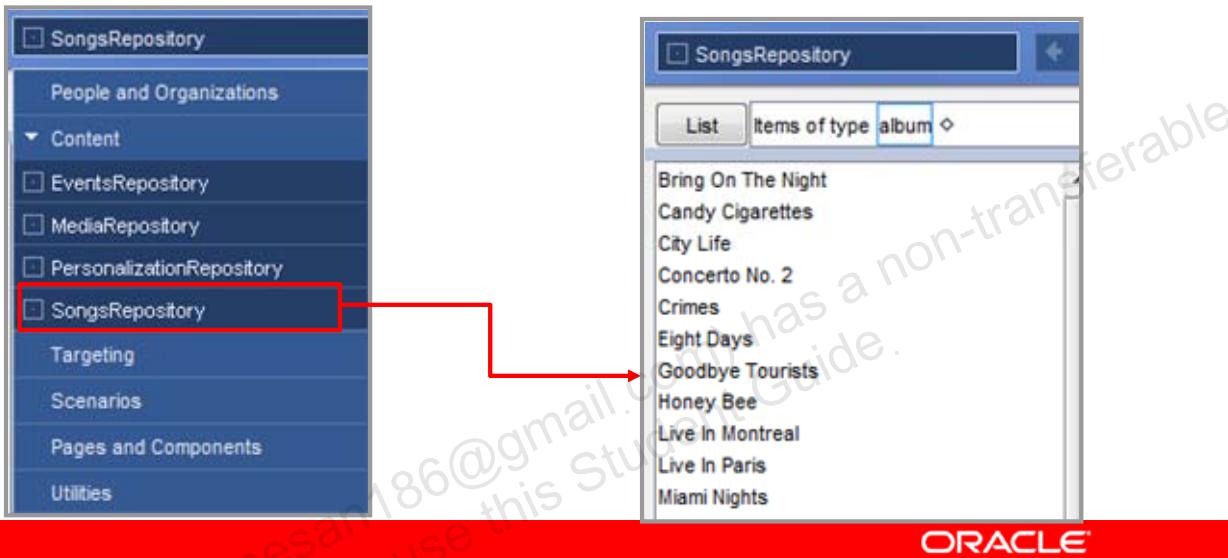
Some useful arguments to this script are:

- -debug: Get debug output from the Repository component
- -export and -import: Store and populate a Repository, respectively

Details can be found in the “Developing and Testing an SQL Repository” chapter of the *ATG Repository Guide*.

## Repositories and the ACC

- To make your repository accessible in the ACC, modify the /atg/registry/ContentRepositories component.
- Add your repository to the initialRepositories property.

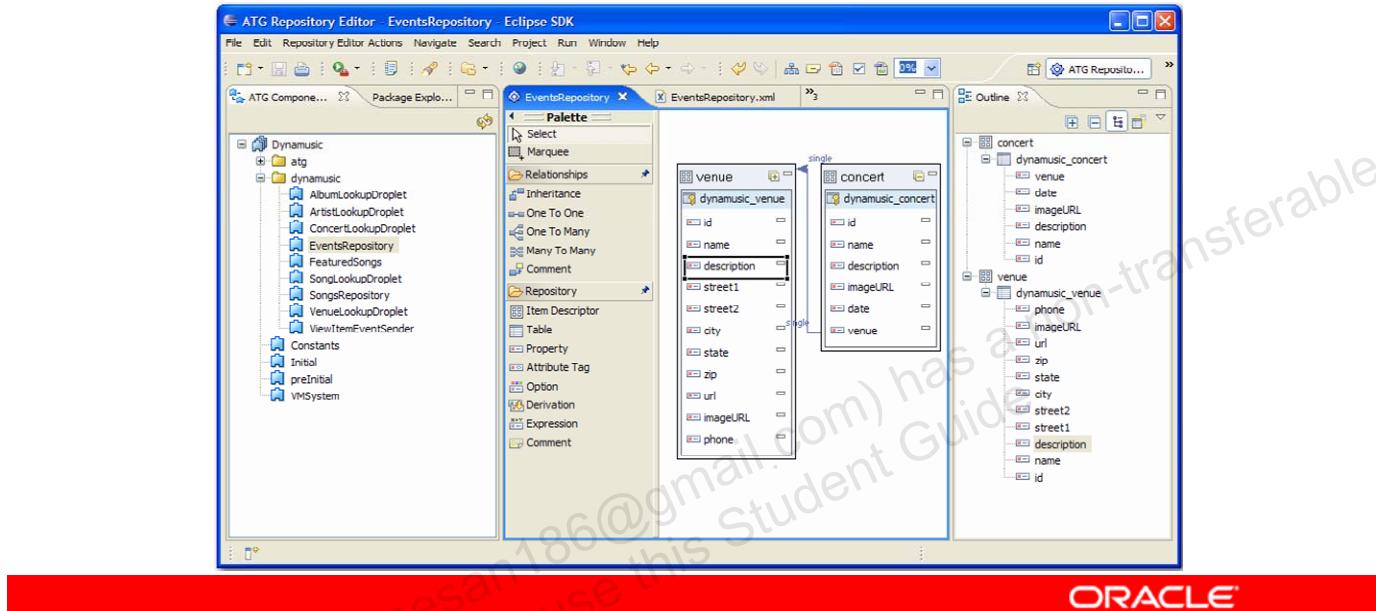


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You must add your repository to the list of repositories in the ContentRepositories.initialRepositories property. When you do this, you ensure that your repository starts when the server starts, and that it appears in the ACC under the Content menu.

## Editing XML

- You can create and edit Repository Definition files by using any XML editor.
- In this course, you use Eclipse to start.



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

XML files are simply text files, and therefore, can be edited with any text editor. However, a template-aware XML-specific text editor, such as the ATG Repository Editor used in your exercises or XML Buddy ([xmlbuddy.org](http://xmlbuddy.org) for more information), can be very helpful by providing you with syntactically correct options, syntax checking, and so on.

**Note:** The ATG Repository Editor plug-in for Eclipse takes advantage of the features of Callisto, an Eclipse add-on. If you install Eclipse Europa or higher, the necessary Callisto features are automatically included.

## SQL Repository Development Checklist

1. Design the data object model.
2. Make sure that the database is running.
3. Create database tables if necessary.
4. Create the GSARespository component.
5. Configure Item Descriptors in XML.
6. Test by using startSQLRepository.
7. Add the repository to  
/atg/registry/ContentRepositories.
8. Restart ATG and the ACC.
9. Review the repository in ACC or in JSPs.



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

## Summary

In this lesson, you should have learned how to:

- Configure a SQL Repository (GSA) component
- Define Repository Items and map their properties to database tables
- Define relationships between Repository Items
- Test and debug a SQL Repository

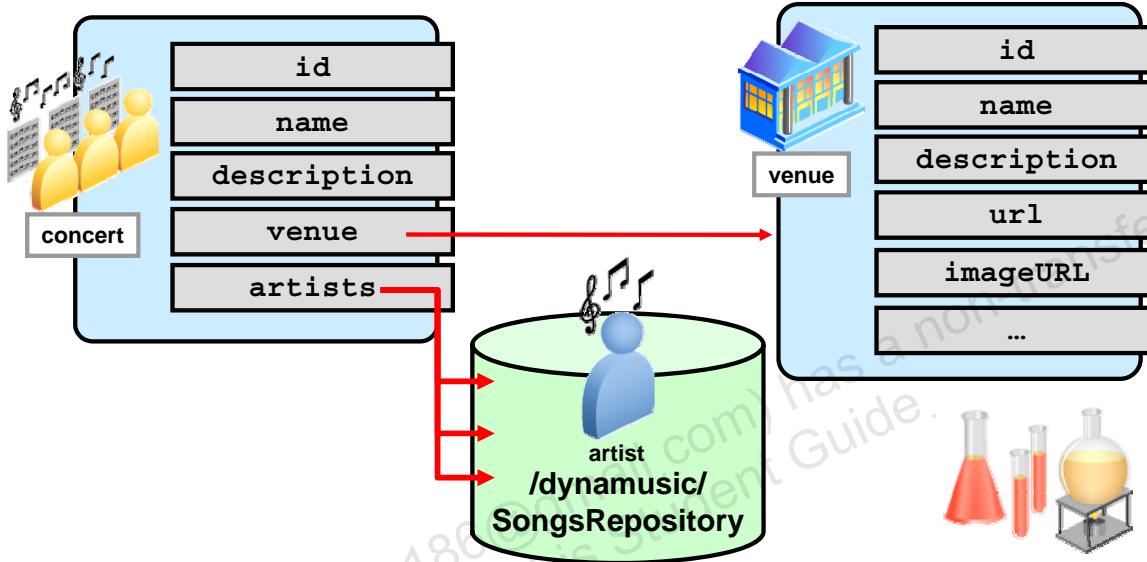
## **Additional Information**

### ***ATG Repository Guide***

- “SQL Repository Overview”
- “SQL Repository Architecture”
- “SQL Repository Data Models”
- “SQL Repository Item Properties”
- “SQL Repository Reference”

## Practice 5 Overview: Configuring SQL Repositories: Practice 5-1

This lesson has two practices. The first practice covers creating, populating, and testing an Events SQL Repository with two item types: **concert** and **venue**.

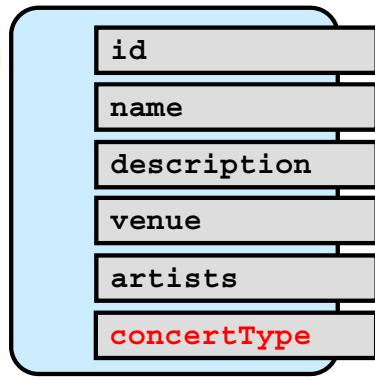


ORACLE

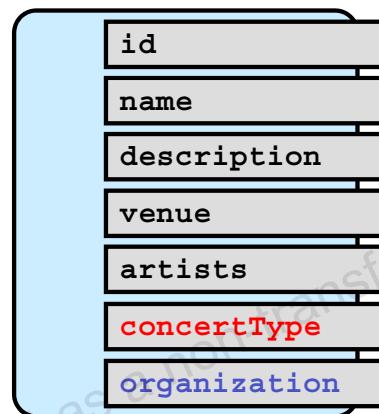
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 5 Overview: Configuring SQL Repositories: Practice 5-2

The second practice covers adding a **concert-fundraiser** subtype for fundraiser concerts.



concert-fundraiser



ORACLE

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

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

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

# Using Repository Form Handlers



ORACLE®

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

# Objectives

After completing this lesson, you should be able to:

- Explain ATG's form-handling capabilities
- Create a form handler by using the `RepositoryFormHandler` class
- Create a form page that allows the user to modify repository data
- Create and use a search form handler to search the repository



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

## Dynamusic Forms

- Dynamusic.com needs a way for users to enter information on the site:
  - Concert and album reviews
  - User registration and profile updates
  - Song and event data (admin users)
  - Discussion group bulletin board postings
- User input needs to be received, validated, and stored in a repository.



ORACLE

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

# ATG Forms

ATG provides a powerful mechanism for handling forms in JSPs, by combining:

- **DSP form tags:** Special versions of the standard HTML form tags that link form fields to component properties
- **ATG Form Handlers:** Components that manage form input and processing



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

In ATG, forms are able to display and set Nucleus component property values. A form can *display* the current value of a Nucleus component property. When a user submits a form, the values the user enters can *set* the value of component properties.

## ATG Form Handlers

Form Handlers are components that can accept input from an HTML form and:

- Validate the input
- Usually pass data to other business-level components to process data (and store the data in a repository)
- Sometimes process the data in the form handler itself (for simple situations)
- Redirect the user to another page based on the results of the form processing



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

ATG Form Handlers are used frequently throughout the ATG product suite. Most Form Handlers in ATG are based on the `atg.droplet.GenericFormHandler` class or one of its subclasses. In this lesson, you learn how to use two of ATG's form handler classes that are specific to repositories. Building custom Form Handlers is covered in the *Extending the ATG Platform* course.

# Road Map

- Repository form handlers
- Search form handlers



ORACLE

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

## RepositoryFormHandler

- ATG provides a specialized class to handle form input to a repository:
  - Supports creating new items, and updating or deleting existing items
  - Handles errors
  - Redirects to specified pages based on the result of the operation
- `RepositoryFormHandler` is a class for component instances configured for repository and Item Type.



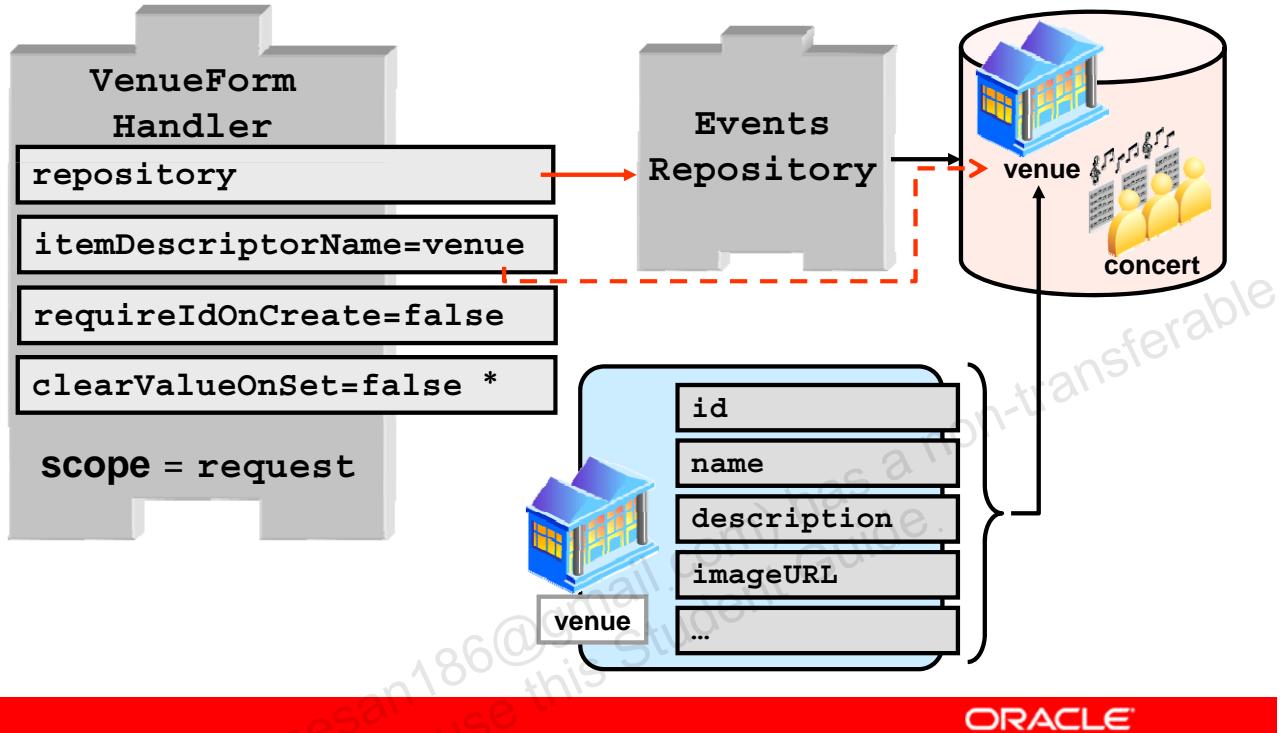
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The full name of the class is `atg.repository.servlet.RepositoryFormHandler`.

This class allows most repository input to be handled without a developer having to write any Java code. Sometimes some additional processing needs to be done beyond what `RepositoryFormHandler` can accomplish, such as setting up relationships between repository items. In that case, `RepositoryFormHandler` can be used as a base class, with a minimal amount of Java supplied by the developer to the subclass.

# Configuring a Repository Form Handler

Create a new component based on the `RepositoryFormHandler` class.



ORACLE

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

Configuring a repository form handler is very similar to configuring a component based on `ItemLookupDroplet`.

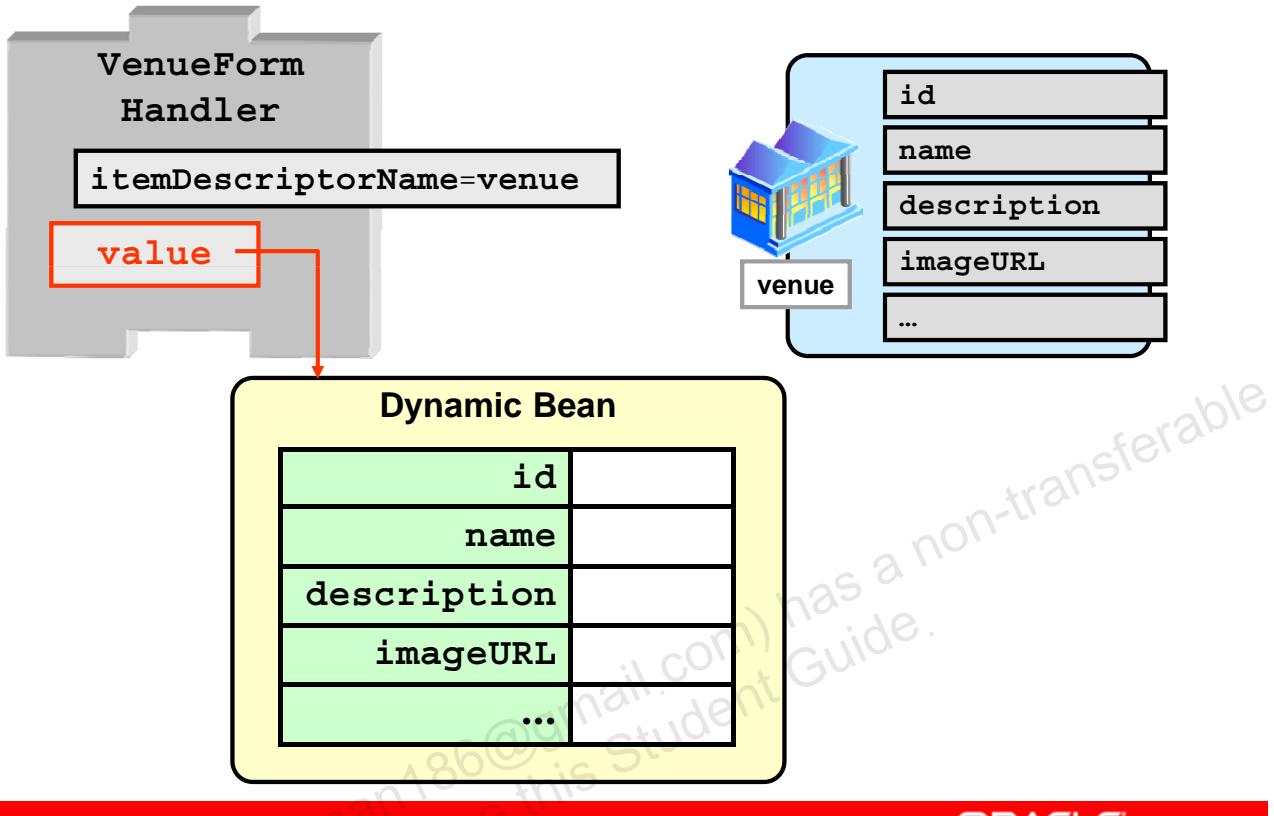
`requireIdOnCreate` (default is `true`) requires that the form supply an ID when creating a new item. If you are relying on the repository (using the ID Generator) or the database to supply IDs, you set this to `false`. Usually, you leave it `true` only if you are using a user-supplied value as the ID for an item. If this property is `true`, and an ID is *not* supplied, the form throws an exception. Handling of exceptions in a form is discussed shortly.

`clearValueOnSet` (default is `true`) specifies whether the value dictionary is cleared and reset from the database when its `repositoryId` property changes. When this property is set to `true`, data in the value dictionary reflects the item's database values. When set to `false`, the value dictionary can retain some data as the `repositoryId` changes from one item to another.

\*If your form handler has session scope, you would use the default setting to clear the values when the `repositoryId` changes.

**Note:** Setting this property to “false” on a request-scoped form handler is one way to make sure that the repository ID gets set before the value is cleared. There are other ways to control this that you will learn about later in the course.

## Handling Dynamic Properties



ORACLE

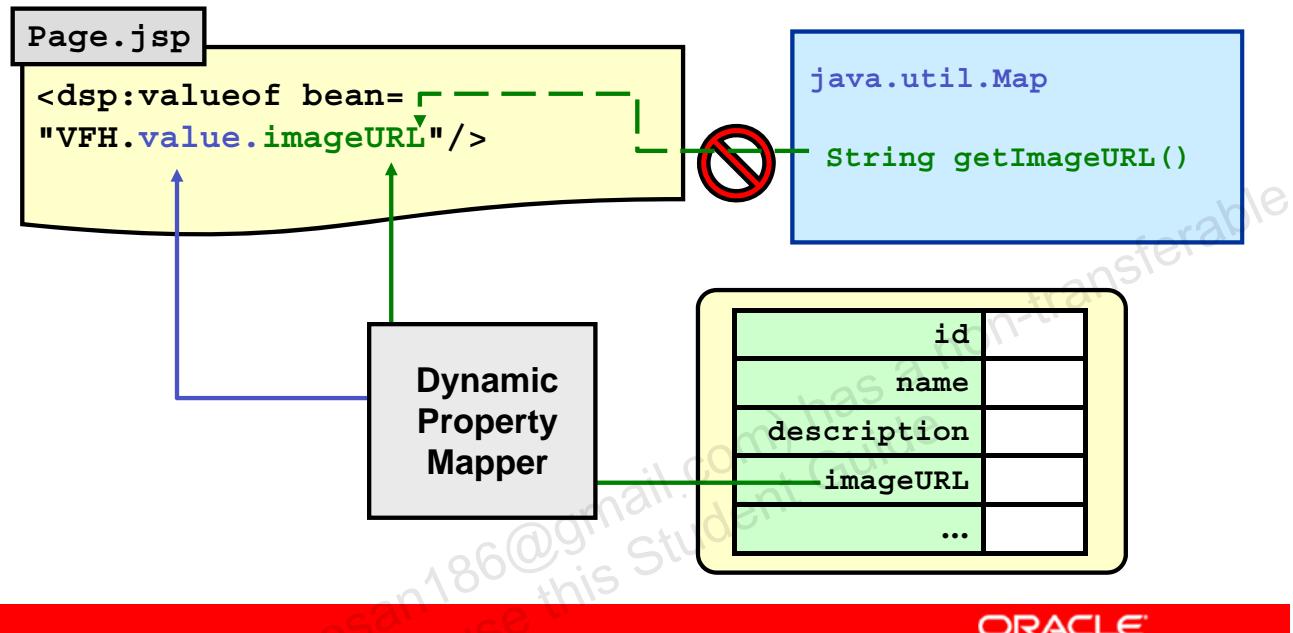
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The RepositoryFormHandler works with any type of repository item, and therefore, cannot have specific properties (get and set methods) corresponding to the properties for any given Item Type. In fact, the form handler does not know until runtime what properties it may be called upon to handle. This problem is solved by the value property of the RepositoryFormHandler, which points to an object of type dictionary, which contains entries for the properties of the Item Type for which the form handler is configured. The Dynamic Beans property mapper enables the entries of the map to be referred to on a page using the component "dot" notation, for example,

<dsp:valueofbean="VenueFormHandler.value.name" />.

## Accessing Dynamic Bean Properties

ATG normally accesses properties by calling the get/set methods of beans, but this will not work for a dictionary.



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

ORACLE

When using ATG tags, such as `dsp:valueof` (and, as you will see shortly, `dsp:input`), the default behavior of the “dot” notation that ATG uses (such as `VenueFormHandler.value.imageUrl`) is to call a property method (such as `value.getImageURL()`).

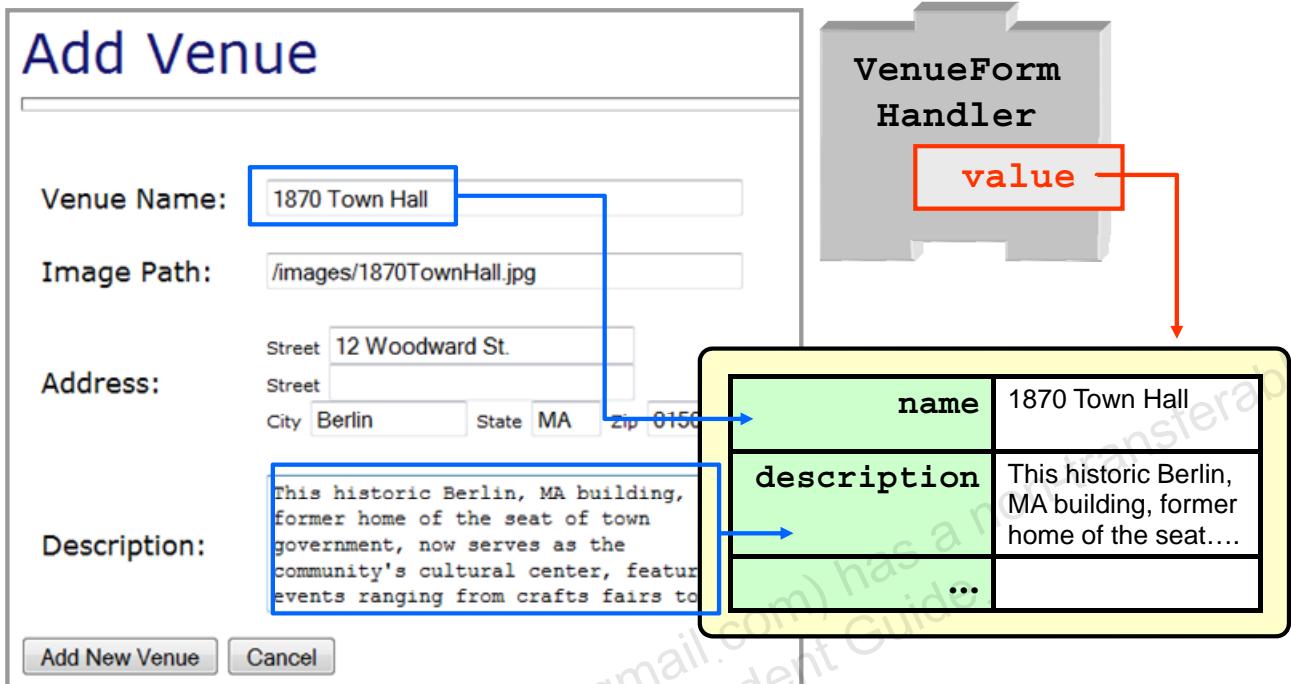
However, for an object that is *not* a JavaBean, such as a map or dictionary, no such method exists. Instead, every Java class that can be used as a Dynamic Bean has a Dynamic Property Mapper registered with ATG that knows the correct method to call to retrieve and set properties.

For example, for a dictionary, the Dynamic Property Mapper translates "value.imageUrl" to `Dictionary.get("imageUrl")`.

You have already seen an example of another type of Dynamic Bean: a `RepositoryItem`. That class also has a registered property mapper, which would translate "venue.imageUrl" into `RepositoryItem.getPropertyValue("imageUrl")`.

You can develop your own Dynamic Beans and register a Dynamic Property Mapper. For details, see the *Programming Guide*, “Nucleus: Organizing JavaBean Components.”

# Using DSP Form Tags



ORACLE®

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

The fields of the form page need to be associated with the properties of the form handler.

## Using DSP Form Tags in the JSP

newVenue.jsp

```
<dsp:form action="newVenue.jsp" method="post">  
...  
Venue Name:  
<dsp:input bean="VenueFormHandler.value.name"  
type="text"/>  
...  
Description:  
<dsp:textarea  
bean="VenueFormHandler.value.description" cols="40"  
rows="4"/>  
...  
</dsp:form>
```

Venue Name:	<input type="text" value="1870 Town Hall"/>
Image Path:	<input type="text" value="/images/1870TownHall.jpg"/>
Address:	<input type="text" value="Street 12 Woodward St."/> <input type="text" value="City Berlin"/> <input type="text" value="State MA"/> <input type="text" value="zip 01503"/>
Description:	<div style="border: 1px solid black; padding: 5px;"><p>This historic Berlin, MA building, former home of the seat of town government, now serves as the community's cultural center, featuring events ranging from crafts fairs to</p></div>

ORACLE

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

A full example of an “update” page is provided in this course’s student files under  
<coursedir>/examples/chapter06/newVenue.jsp.

## DSP Input Tags

- DSP input tags must be used inside a `<dsp:form>` tag.
- DSP form input tags have the same attributes as the corresponding HTML tags *plus* a `bean` attribute:
  - `dsp:input`
  - `dsp:textarea`
  - `dsp:select`
  - `dsp:option`



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

A full list of all the DSP form tags, and their attributes and functions, can be found in Appendix A of the *Page Developer's Guide*.

The following are the input tags:

- `dsp:input`: Has several types, such as `text`, `password`, `checkbox`, `submit`, and `radio`
- `dsp:textarea`: Is a larger text box that allows scrolling and word-wrap
- `dsp:select`: Creates a drop-down list in the form
- `dsp:option`: Is used with the `dsp:select` tag to define options in the drop-down list

(Documentation on the use of HTML input tags can be found in any HTML reference.)

## Default Values

- Property default values are based on the current value.
- Default values can be overridden by the page.
- Syntax depends on the source of the default value.

- Component property:

```
<dsp:input bean="ArtistFormHandler.value.name"  
    beanvalue="Profile.favoriteArtist.name" .../>
```

- Parameter:

```
<dsp:input bean="AlbumFormHandler.value.title"  
    paramvalue="element.title" .../>
```

- Literal:

```
<dsp:input bean="ConcertFormHandler.value.image"  
    value="No image available" .../>
```



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

Regarding the attributes shown in the slide, the `dsp:input` tag works like the `dsp:setvalue` tag. (Note that the `dsp:param` tag is different in that it uses “bean” and “param” rather than “beanvalue” and “paramvalue.”)

If you provide a default value in the input tag (or the current value of the bean is displayed) and the user does not enter a different value, the value displayed in the form will be stored when the user clicks the Submit button.

For example, you have a component with the property `age`, which has a value of 25.

If you specify a value of 30 for `age` in the form tag, the value 30 is displayed.

If the user does *not* enter a value for `age`, the value is set to 30 when the user clicks the Submit button.

## DSP Input Tag Options

- You can mark an input field as “required.”
- An exception will be thrown if the user does not supply a value:

```
<dsp:input bean="VenueFormHandler.value.name"  
required="true"/>
```

- Converters can be used to convert string input to the data type required by the property definition:

```
<dsp:input bean="ConcertFormHandler.value.date"  
date="M/dd/yyyy"/>
```



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

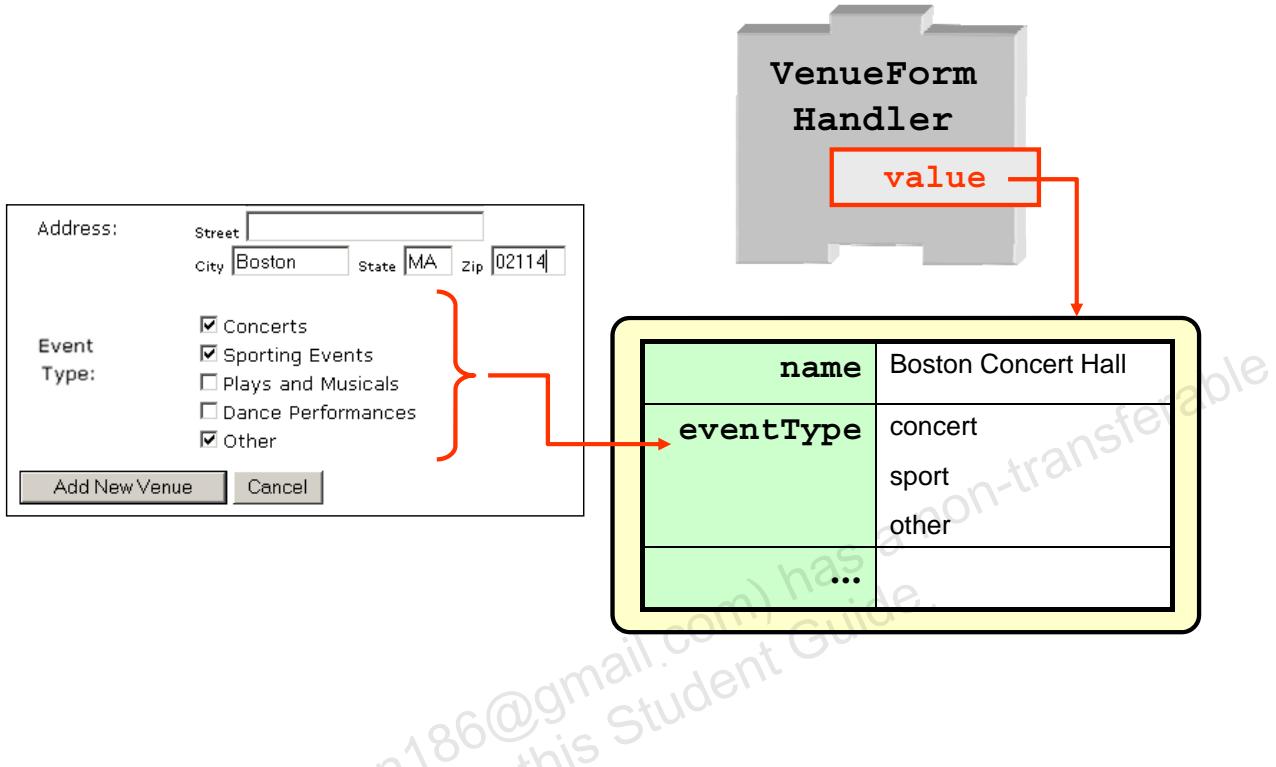
Processing of exceptions (including the one thrown by the `required` attribute) is covered later.

Converters were discussed previously in the lesson titled “Dynamic Pages” in the context of converting property values for display using `<dsp:valueof>`. The same converters work for input in DSP input tags. Converters, including the format for using all the converters, as well as how to create custom converters, are covered in the “Creating Java Server pages” chapter of the *Page Developer’s Guide*. (The “`required`” attribute is technically a converter, so it is covered in the same place in the documentation.)

Converters supplied by ATG include:

- `required`
- `date`
- `mindate`
- `maxdate`
- `number`
- `nullable`
- `currency`
- `creditcard`

## Multivalue Form Input



ORACLE

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

Imagine that you had created a property for venue which was a multiproperty (component-data-type=string, data-type=set, for instance) and you wanted to provide the user a way to select from a list of possible options. (The property might be enumerated or it might not.) In this example, Dynamusic would like to add a property that specifies what sort of events a venue specializes in, and the user of the form can select from any of the available options.

The XML definition for the `eventType` property would look something like this:

```
<property name="eventType"
  category="dynamusic" data-type="set"
  component-data-type="string"
  column-name="event_type"/>
```

## Multivalue Form Input in the JSP

```
<dsp:input type="checkbox"
    bean="VenueFormHandler.value.eventType"
    value="concert" checked="true"/>Concerts<br>
<dsp:input type="checkbox"
    bean="VenueFormHandler.value.eventType"
    value="sport"/>Sporting Events<br>
<dsp:input type="checkbox"
    bean="VenueFormHandler.value.eventType"
    value="theater"/>Plays and Musicals<br>
<dsp:input type="checkbox"
    bean="VenueFormHandler.value.eventType"
    value="dance"/>Dance Performances<br>
<dsp:input type="checkbox"
    bean="VenueFormHandler.value.eventType"
    value="other"/>Other<br>
```

name	Boston Concert Hall
eventType	concert sport other

ORACLE

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

Notice in the code example in the slide that there is a `<dsp:input>` tag for each check box in the list, but that they all refer to the same bean property. This means that *all* the values the user enters will be stored in the multivalue `eventType` property. The actual string stored is specified by the `value` attribute of the tag (such as "theatre"). The string *displayed* is not part of the tag at all (such as "Plays and Musicals").

The "concert" check box is marked `checked="true"` indicating that this value should default to checked. The `value` tag is used differently here than in a text input field; here, it is used to specify the value to add to the multivalue property, rather than a default value.

Another way to allow users to select several items is by using a `<dsp:select>` tag (for a drop-down list) along with `<dsp:option>` tags for each item on the list. The details of that syntax are available in the *Page Developer's Guide*, "Appendix A: DSP Tag Libraries" reference on the `<dsp:select>` tag.

The approach to allow the user to select a single value from a list of options is similar. Instead of check boxes, you would use radio buttons. Or, instead of a *multi* select list, you would use a *single* select list.

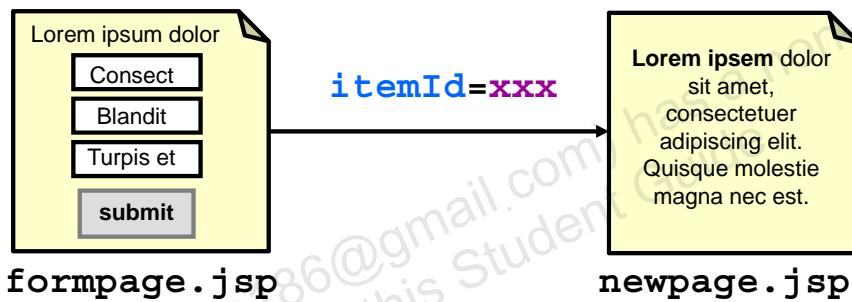
## HTML Hidden Input Fields

HTML hidden form fields allow you to pass a value to the target page without allowing the user to see or change it.

```
<dsp:form action="newpage.jsp">  
  <input type="hidden" name="itemId"  
        value="xxx">...
```

The above code is equivalent to URL:

`newpage.jsp?itemId=xxx`



ORACLE

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

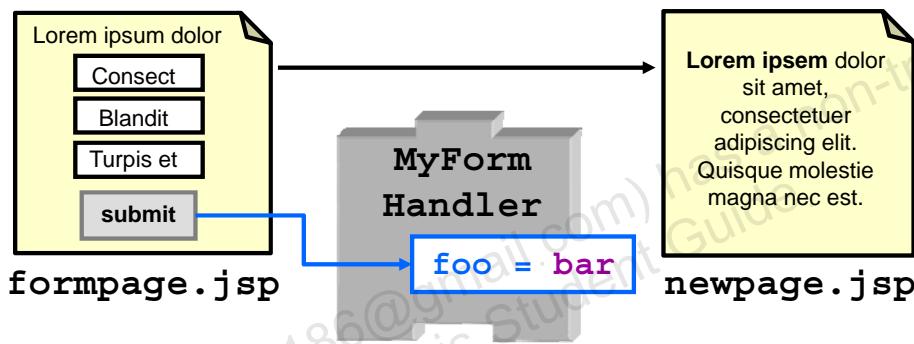
Because this example shows a *plain HTML* input tag instead of a DSP input tag, the value passed must be a literal; only DSP tags support the use of `paramvalue` and `beanvalue`. To use a dynamic value, you would have to get the value of the bean or param yourself, such as:

```
<input type="hidden" name="itemId" value='<dsp:valueof  
param="element.id"/>'>
```

## DSP Hidden Input Fields

DSP hidden form fields allow you to pass a value *to the form handler* without allowing the user to see or change it.

```
<dsp:form action="newpage.jsp">
  <dsp:input type="hidden"
    bean="MyFormHandler.foo" value="bar"
  />...
```



ORACLE

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

DSP hidden input tags do the same things as regular HTML hidden input tags with the added ability to:

- Use a literal, a bean property, or a page parameter as the source of a value (Plain HTML supports only literal values.)
- Set the value in a bean property (most often the form handler) when the form is submitted, in addition to passing a parameter to the target page

Often, you do this to allow the page designer control the behavior of the form handler. If the component being set were session or global scope, you could simply use the `dsp:setvalue` tag to set its properties. However, remember that Form Handlers are most often request scoped; the values need to be set, not as part of the current request (which would happen with the `dsp:setvalue` tag) but as part of the form submission request. That is what `dsp:input` tags do.

Plain HTML input tags and DSP input tags can be mixed and matched freely within a DSP form. If you are not setting a bean property, and just want to pass the value as a parameter, then the regular HTML `<input>` is correct.

Note the slight syntactic difference between HTML input tag (which does not require a slash to close the tag) and DSP input (which, like all JSP tags, does require a closing slash).

Hidden DSP fields are used frequently in ATG, as will be shown in use shortly.

# Quiz

The RepositoryFormHandler class:

- a. Supports only updating or deleting existing items
- b. Handles errors
- c. Redirects to specified pages based on the result of the operation
- d. Is a class for component instances configured for repository and Item Type



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

**Answer: b, c, d**

## Submission Handler Methods

- HTML forms are processed when a form Submit button is clicked.
- `RepositoryFormHandler` supplies several methods that can be called on submission to process the data in the form:
  - `create`: Creates a new item based on the form input
  - `update`: Updates the current repository item based on form input
  - `delete`: Deletes the current repository item
  - `cancel`: Cancels the form in progress



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

The actual method names that support this functionality are:

- `handleCreate()`
- `handleUpdate()`
- `handleDelete()`
- `handleCancel()`

## Submission Handler Methods in the JSP

```
<dsp:form action="newVenue.jsp" method="post">
...
<dsp:input bean="VenueFormHandler.create"
  type="Submit" value="Add New Venue"/>
<dsp:input bean="VenueFormHandler.cancel"
  type="Submit" value="Cancel"/>
...
</dsp:form>
```

ORACLE

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

The “magic” that associates the property “create” with the method of the form handler that does the actual creation is explained in more detail in the *Extending the ATG Platform* course.

## Page Redirection

- By default, when the form is submitted, the form's "action" page will be displayed:  
`<dsp:form action="newVenue.jsp" method="post">`
- RepositoryFormHandler lets you specify a different page to redirect to based on the results of the submission:

```
<dsp:form action="newVenue.jsp" method="post">
...
<dsp:input
    bean="VenueFormHandler.createSuccessURL"
    type="hidden" value="venues.jsp"/>
...
</dsp:form>
```



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

Recall from previous slides that DSP hidden input tags let the page set values on the form handler. This feature is used frequently in setting redirection URLs; that way the page designer can set the values when the page is submitted.

## Page Redirection Properties

RepositoryFormHandler has both success and error redirection properties for each submission handler:

- createSuccessURL
- createErrorURL
- updateSuccessURL
- updateErrorURL
- deleteSuccessURL
- deleteErrorURL
- cancelURL



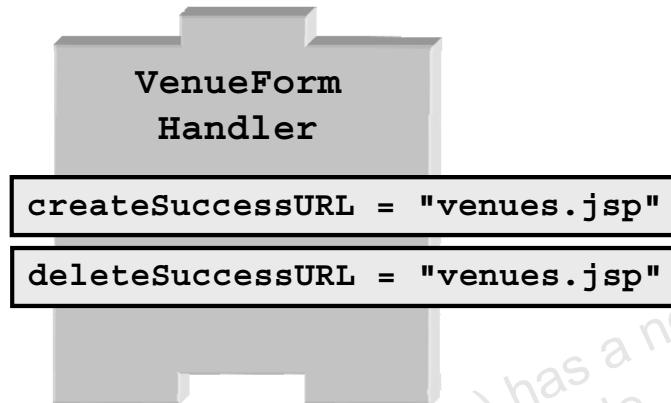
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If any of the properties listed in the slide are not set, if the associated condition occurs, no redirection will occur and the “action” page will be displayed.

Note that each type of form action has associated error and success URLs except for cancel. Cancel has only a single URL because cancel cannot fail or succeed.

## Setting Redirect Properties

In many cases, best practice is to set redirect URLs in the .properties file.



ORACLE

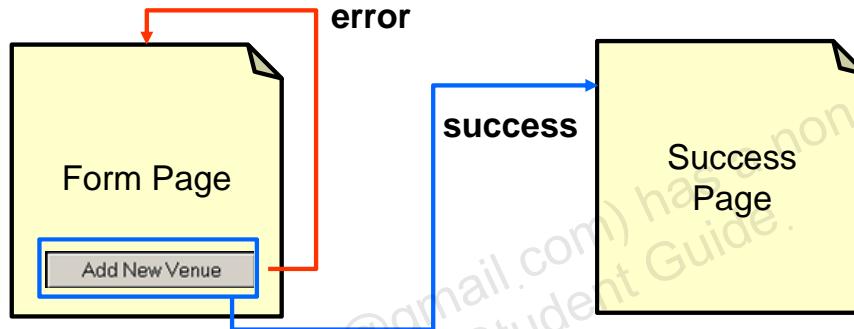
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Setting the redirect properties in the .properties file allows all redirect configuration to be done in one place, with less clutter in the JSPs. However, note that the URLs shown in the slide above are completely hard coded, and that updateSuccessURL is not shown. If a redirect URL needs to be resolved dynamically (for example, if parameters need to be appended), it may be better to set it in the JSP. You will see an example of this for updateSuccessURL later in this lesson. If you want updateSuccessURL to redirect to the venueDetails.jsp, showing the venue that has just been updated, then you need to dynamically append a parameter with the ID of the particular venue that has just been updated.

## Common Form Page Design

One common approach to form page redirection is to set the action page back to the form page.

- Process errors on the form page itself, if necessary.
- Redirect on success by setting the success URL property.



ORACLE

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

## A Common Form Page Design: JSP Expression

Rather than hard-coding in the current page as the “action” of a form, use a JSP expression:

```
<dsp:form method="post"  
action="<% =request.getRequestURI () %>">
```



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

Every JSP has access to a `request` variable, which points to an object representing the HTTP request for the page. This object implements the interface `javax.servlet.http.HttpServletRequest`. For more information, see the *API Reference*.

## Quiz

Which of the following are methods that can be called on submission to process the data in a form?

- a. delete
- b. add
- c. cancel
- d. update



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

**Answer: a, c, d**

# Error Handling

**RepositoryFormHandler** has properties to help manage form errors:

- `formError` (boolean): True if an error occurred when processing the form
- `formExceptions`: A vector of all exceptions that occurred
- `propertyExceptions`: A dictionary that maps repository item properties to exceptions that occurred while processing that property



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

The `formExceptions` property is typically used to display error messages at the top of the form. The `propertyExceptions` property can be used to display error messages on a per-field basis (for example, by changing the color of the font on the label of the fields in error).

## Error Handling: ErrorMessageForEach

You can use the **ErrorMessageForEach** Servlet Bean to loop through and display the error messages that are generated by a form:

```
<dsp:droplet
    name="/atg/dynamo/droplet/ErrorMessageForEach"
    >
    <dsp:oparam name="output">
        <dsp:valueof param="message"/> <br>
    </dsp:oparam>
</dsp:droplet>
```



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

The most frequently used parameters of the **ErrorMessageForEach** Servlet Bean are:

- Output params:
  - Message: The message for the current exception
  - Property: The property the form handler was trying to set when the exception occurred
- oparms
  - Output: Rendered once for each exception in the list
  - outputStart, outputEnd, empty: Just like the **ForEach** droplet

Details on the rest of the parameters, which are primarily for use with custom Form Handlers or internationalization, are described in Appendix B of the *Page Developer's Guide*.

Note that some standard error messages are defined in `/atg/droplet/ErrorMessageResources.properties`, which is provided in `atgdir/DAS/lib/resources.jar`. These can be overridden in your module by editing and placing `/atg/droplet/ErrorMessageResources.properties` in your module's classpath folder.

## ErrorMessageForEach: Example

The screenshot shows two instances of a JSP form for adding a venue. The first instance has a red border around the 'Venue Name:' field, which is empty. A red arrow points from this field to a validation message box containing the text: "• You must supply a value for the 'name' field of this form." The second instance is a redisplayed version of the form after an error. It shows the same fields, but the 'Venue Name:' field now contains the message: "This venue has no name, so it can't really exist anyway, right?" This message is displayed in a scrollable text area.

Venue Name:

Image Path: images/NoSuchVenue.jpg

Address:

Street   
Street   
City  State

Description:

This venue has no name, so it can't really exist anyway, right?

Add New Venue Cancel

Venue Name:

Image Path: images/NoSuchVenue.jpg

Address:

Street   
Street   
City  State  Zip

Description:

This venue has no name, so it can't really exist anyway, right?

Add New Venue Cancel

ORACLE

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

Because the name field was marked required, if the user does not enter a value, an exception will be thrown. In the case of addVenue.jsp, the "action" page is addVenue.jsp and no createErrorURL was set; therefore, the page is redisplayed. The second time the page was displayed, there were errors present; so the ErrorMessageForEach Servlet Bean displayed them (in this example, only one.)

## Updating Existing Repository Items

The previous example showed the creation of a new item; updating an item requires three extra steps.

1. Populate the form with the existing data.
2. Set the item ID for the item to update on the form handler.
3. Pass the item ID to the “action” page.



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

The same steps apply when deleting an existing repository item.

A full example of an “update” page is provided in examples/chapter06/editVenue.jsp.

## Request-Spaced Form Handlers

request #1: editVenue.jsp

**Edit Venue**

Venue Name:

Image Path:

Address:

Street   
Street   
City  State  Zip

Description:

request #2: success.jsp

ORACLE

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

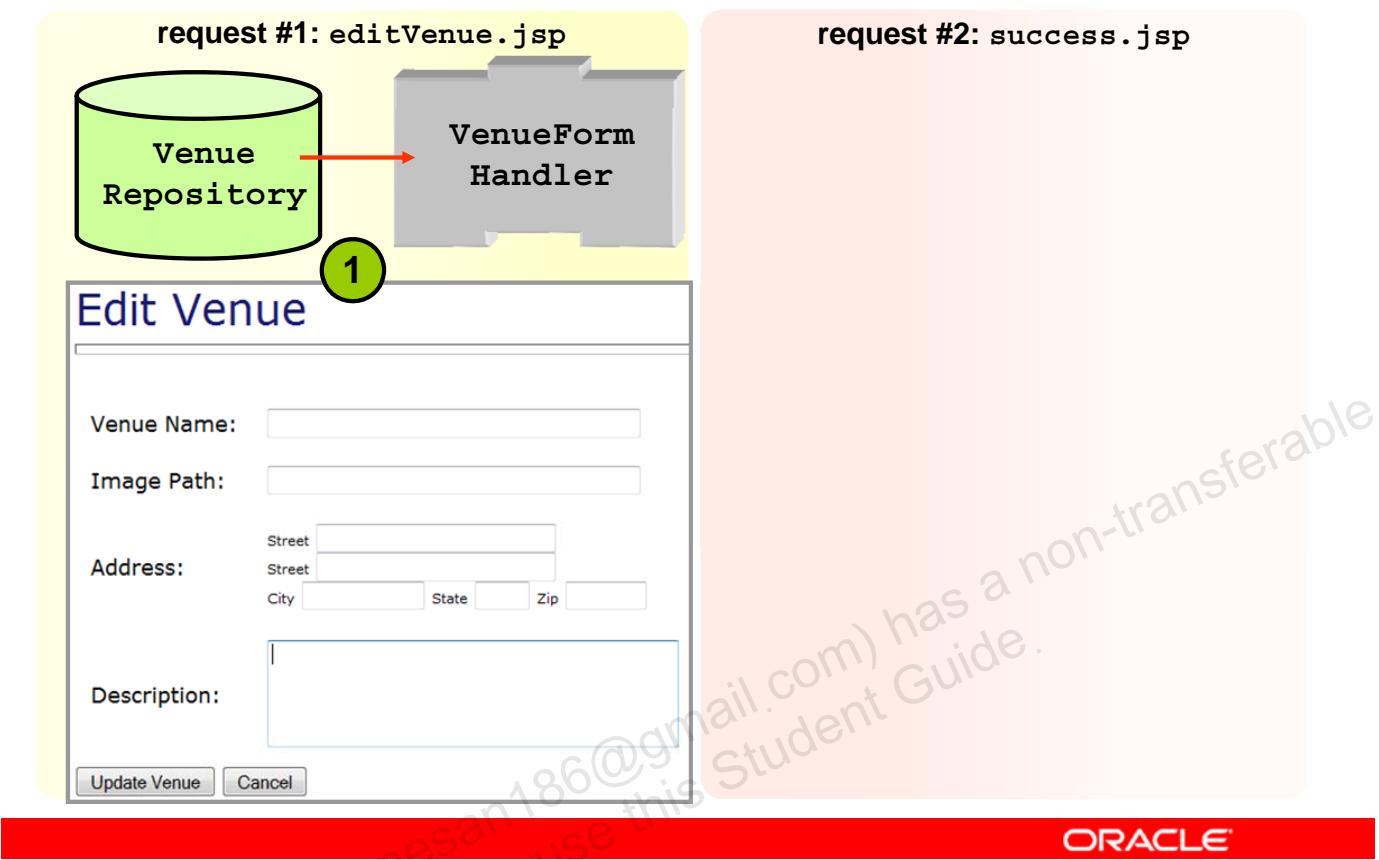
To explain how updating values works, it is important to understand how form submission and handling occurs, and how request-scoped Form Handlers work.

Recall that a request-scoped component exists only for a single page request. When the page finishes being rendered, that instance of the component goes out of scope and ceases to be. In the case of a request-scoped form handler component, two instances of the component are created. The first is used to populate the fields of the form, if appropriate. The second is created when the Submit button is clicked, and receives the field values entered by the user, and on which the `update` method gets called that processes those values and stores them in the repository.

So the order of events is:

1. The user makes a request for a page such as `editVenue.jsp`. During the processing of the page request, an instance of `VenueFormHandler` is created as part of the page request.
2. The form handler is populated with data from the repository, and the page uses that data to display on the page.
3. After the page is displayed, the form handler component goes out of scope and ceases to exist.
4. The user can then interact with the page, such as entering new or changed information, and then clicks the Submit button.

## Request-SScoped Form Handlers

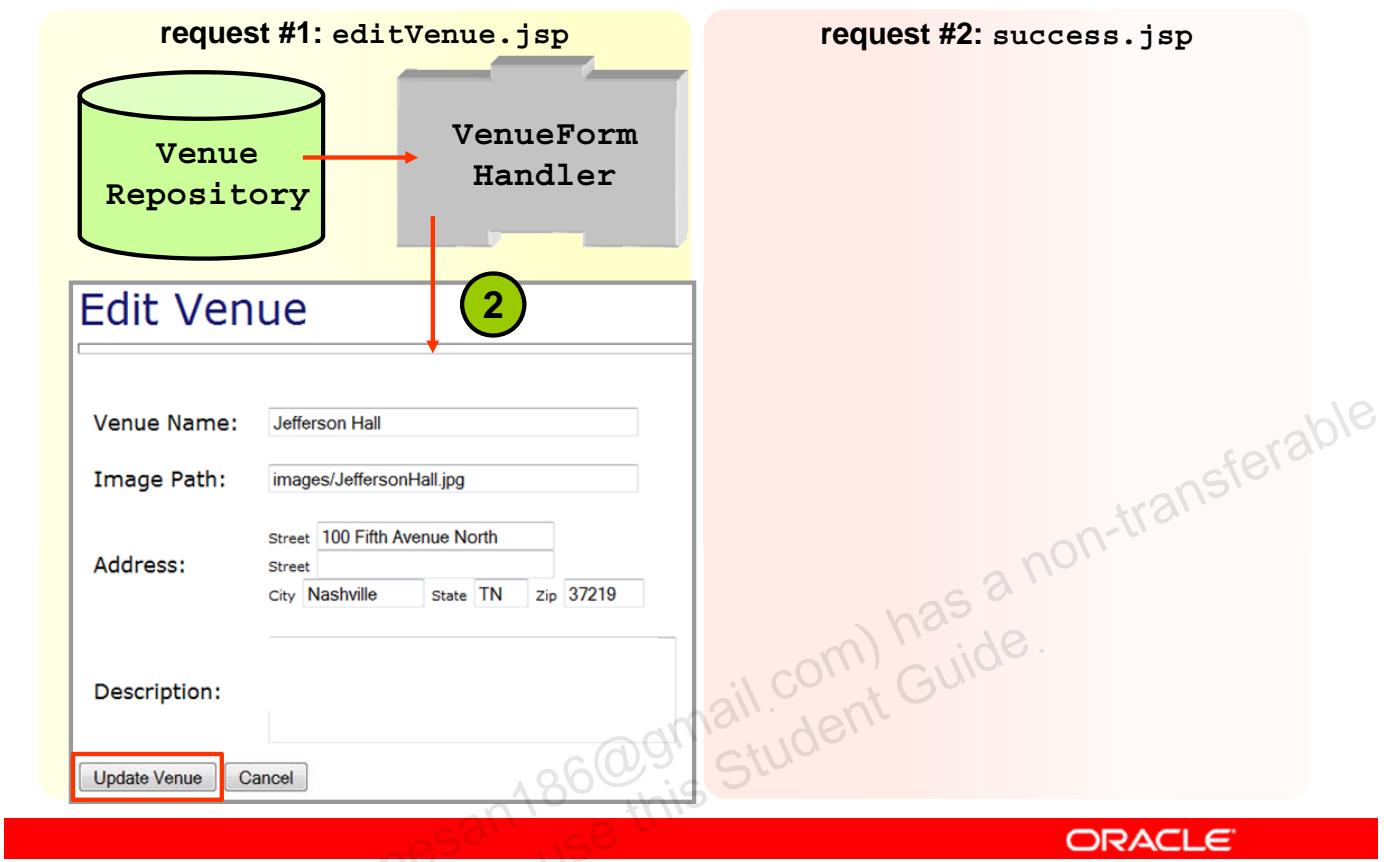


ORACLE®

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

5. This generates a second page request (for example, to a `success.jsp` page). During the processing of that request, a second instance of `VenueFormHandler` is created. The values the user entered on the page (as well as the default values populated during the page request) are passed to the form handler, and the submit method (`handleUpdate()`) is called, which stores the changed values back to the repository.
6. After the second page is displayed, the form handler again goes out of scope, because it is no longer needed.

## Request-SScoped Form Handlers



ORACLE®

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

## Request-Scope Form Handlers

request #1: editVenue.jsp

**Edit Venue**

Venue Name:

Image Path:

Address:

Street:   
Street:   
City  State  Zip

3 Description:

request #2: success.jsp

ORACLE

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

## Request-Scope Form Handlers

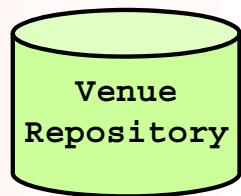
request #1: editVenue.jsp

Edit Venue

Venue Name:	Jefferson Hall
Image Path:	images/JeffersonHall.jpg
Address:	Street: 100 Fifth Avenue North Street: City: Nashville State: TN Zip: 37219
Description:	Jefferson Hall first opened its doors in 1892 as a vision of Captain Jefferson J. Jefferson ...

request #2: success.jsp

VenueForm Handler



ORACLE

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

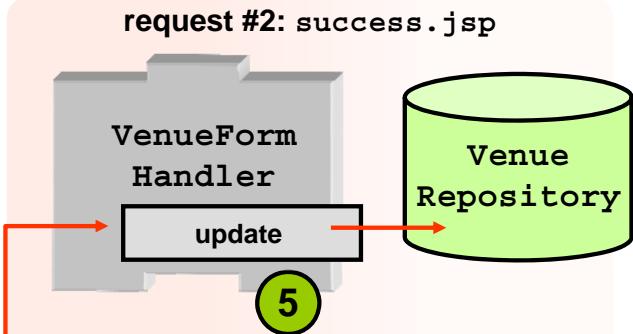
## Request-SScoped Form Handlers

request #1: editVenue.jsp

Edit Venue

Venue Name:	Jefferson Hall
Image Path:	images/JeffersonHall.jpg
Address:	Street: 100 Fifth Avenue North Street: <input type="text"/> City: Nashville State: TN Zip: 37219
Description:	Jefferson Hall first opened its doors in 1892 as a vision of Captain Jefferson J. Jefferson ...

request #2: success.jsp



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

ORACLE

## Request-Scope Form Handlers

request #1: editVenue.jsp

**Edit Venue**

Venue Name:

Image Path:

Address:

Street:   
Street:   
City  State  zip

Description:

request #2: success.jsp

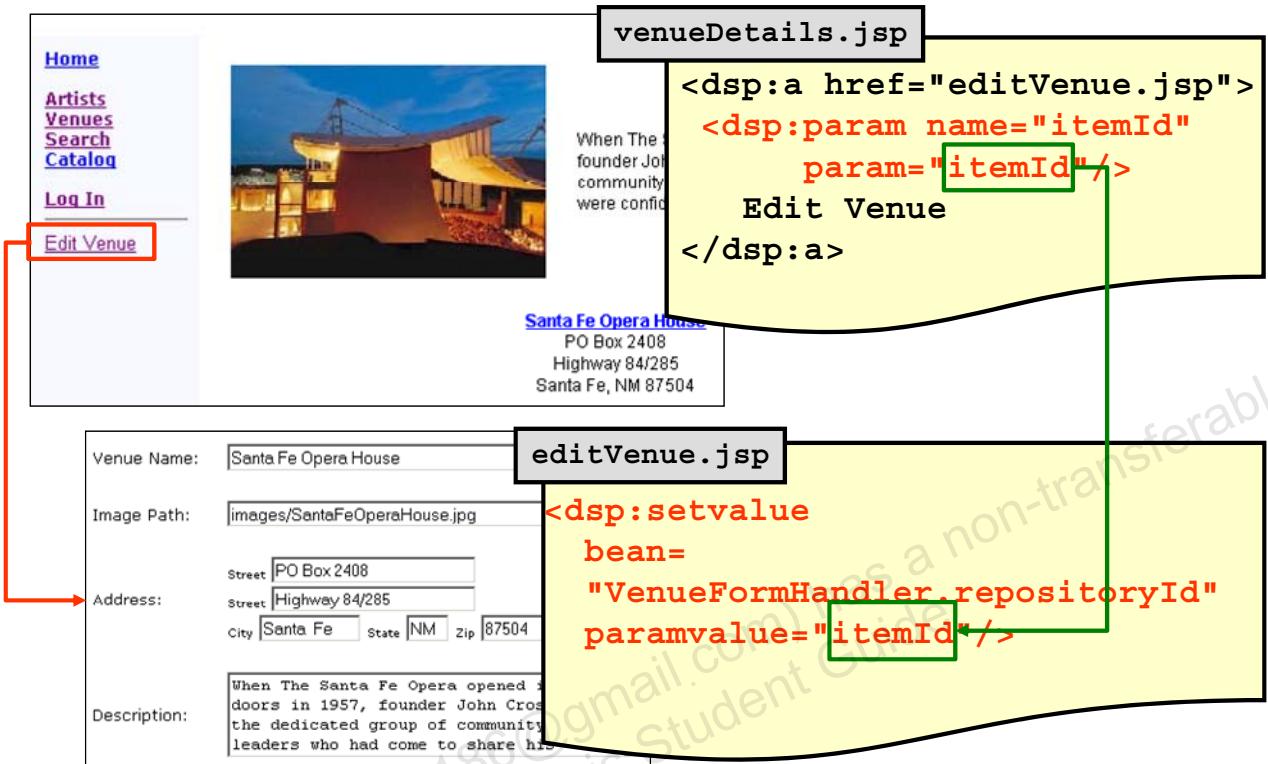
6



ORACLE

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

# Populating a RepositoryFormHandler



ORACLE

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

To update or delete an existing repository item, the form handler has to know which item to work with. This is accomplished by setting the form handler's `repositoryId` property to the item's ID; the repository form handler will then automatically populate the value dictionary with the values from that repository item. The resulting values will be displayed in the form (unless overridden by values set in the input tags).

(You can disable populating the value dictionary based on the repository ID by setting `RepositoryFormHandler.extractDefaultValuesFromItem=false`.)

## Setting the Item ID in the Form Handler

editVenue.jsp

```
<dsp:setvalue  
    bean="VenueFormHandler.repositoryId"  
    paramvalue="itemId"/>  
  
<dsp:form action="<% =request.getRequestURI() %>"  
    method="post">  
    ...  
    <dsp:input bean="VenueFormHandler.repositoryId"  
        type="hidden" paramvalue="itemId"/>  
    ...  
</dsp:form>
```



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

When the user submits the form, the form handler needs to know which item in the repository to update. You might think, based on the previous slide, that because this page displayed the first time and the form handler pre-populated the values, it should already know what the item ID was. It is important to remember that this is a *request-scoped* component. The page that displayed the values had a temporary instance of the component to supply the pre-populated values. When the form is submitted, a *new* instance of the form handler will be created, and by default, that does not know what repository item the values entered are associated with. Using the `dsp:input` tag here sets the `repositoryId` property of the *new* instance of the form handler to the correct value, so that when the `handleUpdate()` method is called, it will know which repository item to update (`type="hidden"` because you probably do not want to expose the repository ID of the item to the user, nor allow them to change it).

## Passing the Item ID to the Action Page

editVenue.jsp

```
<dsp:setvalue  
    bean="VenueFormHandler.repositoryId"  
    paramvalue="itemId"/>  
  
<dsp:form action="<%=>request.getRequestURI()%>"  
method="post">  
...  
    <dsp:input bean="VenueFormHandler.repositoryId"  
    type="hidden" paramvalue="itemId"  
    name="itemId"/>  
...  
</dsp:form>
```



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

If an error occurs when the form is submitted, the page will be re-displayed. But recall that `editVenue.jsp` requires an input parameter of `itemId`, so that it can 1) pre-populate the displayed values, and 2) tell the form handler which item to update. When the page is re-displayed, by default, it will no longer have access to the `itemId` parameter that was passed in the first time. So you will need to pass the parameter again. The easiest way to do that is to have a “hidden” input with a “name” value which matches the parameter value, because HTTP passes form data as parameters using the field name. Because in the previous slide the page was already setting `itemId` in a hidden tag, all that is needed here is to add a “name” attribute to the hidden field.

Another way to look at this is: If the form is displaying venue item 20100, and an error occurs on the update submission, the page will redirect to `editVenue.jsp?itemId=20100`.

Notice the use of the `paramvalue` attribute here. This enables you to pass a value to the form handler using a parameter as the value source. Similarly, `beanvalue` could have been used here. Recall that the equivalent for the `dsp:param` tag would be the `param` and `bean` attributes.

## Passing the Item ID to a Redirect Page: **dsp:getvalueof**

For redirect URLs that require parameters, one technique is to use **dsp:getvalueof**:

```
<dsp:form action="<%="request.getRequestURI ()%>"  
method="post">  
...  
<dsp:getvalueof param="itemId" id="vId">  
  <dsp:input type="hidden"  
    bean="VenueFormHandler.updateSuccessURL"  
    value='<%="venueDetails.jsp?itemId=" + vId %>' />  
</dsp:getvalueof>  
...  
</dsp:form>
```



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

The technique shown in the previous slide (a hidden input with a name= “attribute”) works only when a user redirects to the form’s action page. Redirect URLs are different because there is no default processing that appends parameters to them.

You may recall from an earlier lesson that the DSP tag library has a limitation on nested tag attributes, and that one workaround is to use the **dsp:getvalueof** tag to assign the value you want to access to a local Java variable name, and then use a JSP Expression to access the value.

## Passing the Item ID to a Redirect Page: Format droplet

Another technique is to use the Format droplet:

```
<dsp:droplet name="/atg/dynamo/droplet/Format">
  <dsp:param name="format"
    value="venueDetails.jsp?itemId={itemId}" />
  <dsp:oparam name="output">
    <dsp:input type="hidden"
      bean="VenueFormHandler.updateSuccessURL"
      paramvalue="message"/>
  </dsp:oparam>
</dsp:droplet>
```



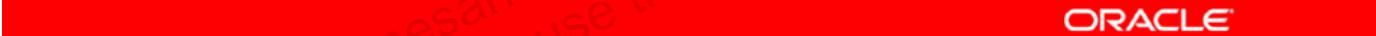
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For a complete description of the Format droplet, see the *ATG Page Developer's Guide*, *Appendix B: ATG Servlet Beans*.

As in the previous slide, the JSP snippet example in the slide assumes there is a parameter called `itemId`. It is delimited by the braces in the example.

## Advanced Features

- All examples shown here are simple item properties.
- The `RepositoryFormHandler` can also handle more complex operations including:
  - Manipulating multivalue properties (adding, removing, and overriding)
  - Managing item relationships
- The advanced `RepositoryFormHandler` functionality is detailed in the *Page Developer's Guide*.



ORACLE

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

# Quiz

Request Form Handlers can be used to load default values on a page in addition to containing values that need updating.

- a. True
- b. False



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

**Answer:** a

# Road Map

- Repository form handlers
- Search form handlers



ORACLE

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

## Search Form Handlers

Search Forms allow users to search for repository items that match criteria.

The screenshot shows a search interface and its results. At the top, there is a search bar with the placeholder "Search for concerts:" followed by a text input field containing the word "live". To the right of the input field is a "Search" button. Below the search bar is a section titled "Concert Search Results". Inside this section, a message says "The following events match your criteria:" followed by a bulleted list of concert names, each with a blue link underneath. The list includes: "Travis B. Live", "Autumn and Travis, Live 2011", "Dexter Bluetone Live", "Kiki Shoppington Live", "Travis B. Live", "Dexter Bluetone Live", "Kiki Shoppington Live", and "Travis B. Live".

Search for concerts: live

Search

### Concert Search Results

The following events match your criteria:

- [Travis B. Live](#)
- [Autumn and Travis, Live 2011](#)
- [Dexter Bluetone Live](#)
- [Kiki Shoppington Live](#)
- [Travis B. Live](#)
- [Dexter Bluetone Live](#)
- [Kiki Shoppington Live](#)
- [Travis B. Live](#)

ORACLE

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

## Using SearchFormHandler

SearchFormHandler supports several kinds of searches:

- **Keyword:** *Simple string comparison (case-sensitive)*
- **Text:** Full text search including wildcards
- **Advanced:** Search for values for different properties, including ranges of values
- **Hierarchical:** Search for items in a hierarchy



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

The full class name for this class is `atg.repository.servlet.SearchFormHandler`.

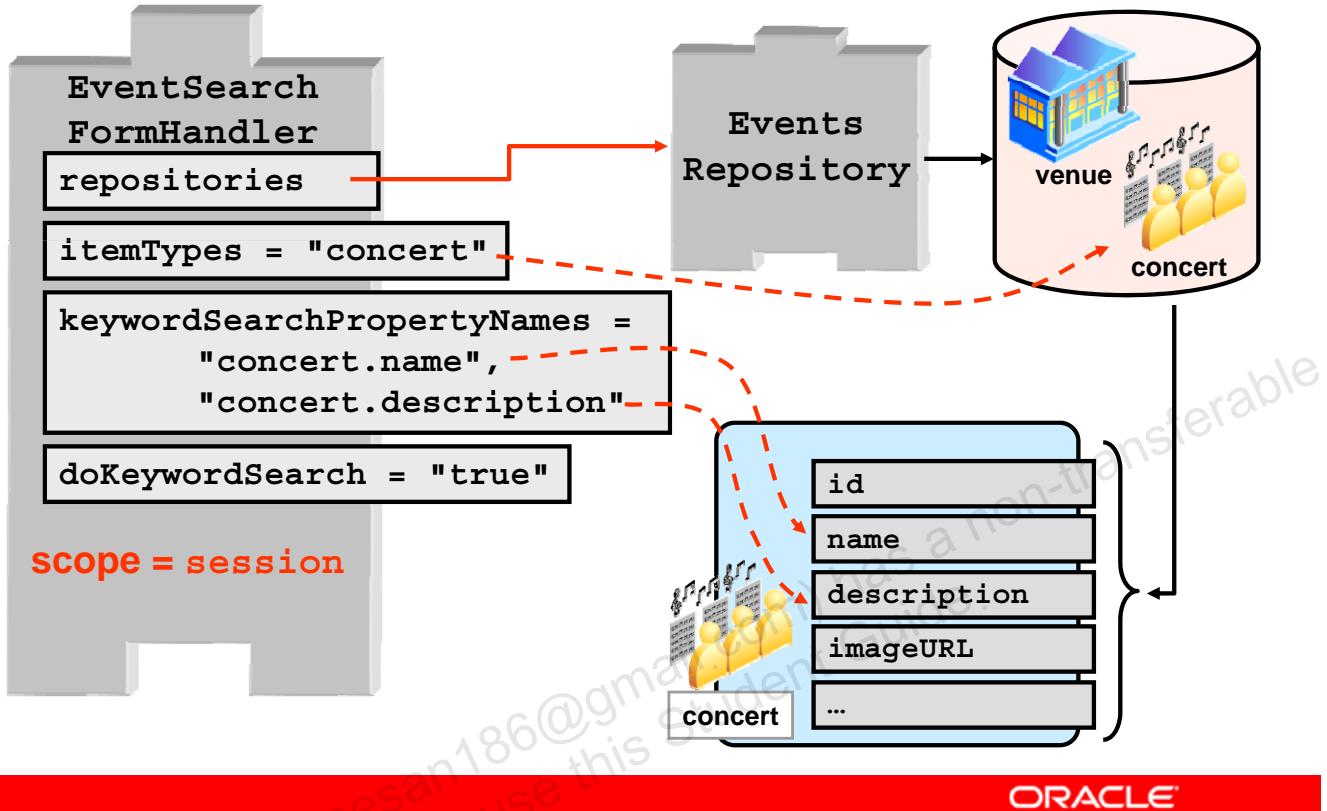
Text and keyword searches are very similar; the primary differences are that text searches require text searching to be enabled on the repository, and uses the text search engine supplied by the database, whereas keyword searches are performed using basic string comparison. Text searches support wildcards.

Advanced searches are useful when searching items that meet very specific criteria in a very large data set. This sort of searching allows the users to specify different criteria for different properties, and also to specify ranges of acceptable values. For instance, the user might search for songs with criteria: `artist=Travis B., playtime between 1 minute and 3 minutes, genre=pop and recording date before 7/2/1985`.

Hierarchical searches are primarily for use by ATG Commerce applications, which arrange product information into a hierarchy of categories.

This course will focus only on keyword searches. Information about using the other types of searching is in the *ATG Page Developer's Guide* under "Search Forms."

# Configuring a Keyword Search Component



ORACLE

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

`EventSearchFormHandler` (or any keyword search form handler) has the following properties that need to be configured

- `repositories`: This is a multivalue property, because the component can search across multiple repositories. (The example shown searches only one repository.)
- `itemTypes`: Which Item Types should be searched for; this is a multivalue property because the component can search for multiple Item Types (although this example searches only for one). If multiple Item Types are searched, each must exist in all repositories searched. Search results of the same item from multiple repositories are combined.
- `keywordSearchPropertyNames`: This value indicates which properties in which Item Types should be searched. If this value is not set, then all “string” type properties of the items are searched.
- `doKeyWordSearch`: Set to `true` to do a keyword search.

One important difference between the `SearchFormHandler` and the `RepositoryFormHandler` (and most other Form Handlers) is that it is designed to be `session` scoped, rather than `request` scoped. This allows the user to return to the results page and have their search results remain available throughout their session.

## Using SearchFormHandler

Search for concerts:

searchevents.jsp

```
<dsp:form action="<% =request.getRequestURI() %>"  
method="post">  
  
    Search for concerts:  
  
    <dsp:input type="text"  
        bean="EventSearchFormHandler.keywordInput"/>  
  
    <dsp:input bean="EventSearchFormHandler.successURL"  
        type="hidden" value="eventSearchResults.jsp"/>  
  
    <dsp:input bean="EventSearchFormHandler.search"  
        type="submit" value="Search"/>  
  
</dsp:form>
```

ORACLE

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

There are two steps to using a search form handler: set up the search and display the results. In the example in the slide, these steps occur on different pages, but they are often combined into a single page.

This slide shows the first page, where the search is set up. On the completion of a successful search, the form handler will redirect to `successURL`, in this case, a separate page called `eventSearchResults.jsp` (shown in the next slide).

This example can be found in the course folder for this course, under `<coursedir>/examples/chapter06/searchevents.jsp`.

The user input text field is attached to the form handler's `keywordInput` property.

The Submit button is attached to the `handleSearch()` method.

Not shown here is the error-handling code, which is identical to how the `RepositoryFormHandler` does error handling, using the `ErrorMessageForEach Servlet Bean`.

## Using SearchFormHandler: Results

eventSearchResults.jsp

```
<p>The following events match your criteria: <p>
<dsp:droplet name="ForEach">
<dsp:param bean="EventSearchFormHandler.searchResults"
  name="array"/>
<dsp:oparam name="output">
<li>
  <dsp:a href="concertDetails.jsp">
    <dsp:param name="itemId" param="element.id"/>
    <dsp:valueof param="element.name"/>
  </dsp:a>
</dsp:oparam>
<dsp:oparam name="empty">
  Sorry, no concerts matched
  your search criteria.
</dsp:oparam>
</dsp:droplet>
```

### Concert Search

The following events match your criteria

- [Travis B. Live](#)
- [Autumn and Travis, Live 2011](#)
- [Dexter Bluetone Live](#)
- [Kiki Shonnington Live](#)

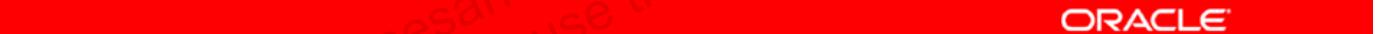
ORACLE

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

After the search is completed, the results are stored as an array in the `searchResults` property of the search form handler. You can use the `ForEach` or `Range` droplet to loop through the array and display the output. In this example, in the `output`, a `dsp:a` tag is used so that each concert name will appear as a link to the `concertDetails` page, passing the ID for the concert.

# Search

- The `SearchFormHandler` component provides very basic searching capabilities using string comparison (keyword) or database text searching (text).
- The Oracle Endeca Commerce product provides more advanced searching such as:
  - Natural language processing
  - Contextual and concept searches
  - Relevance ranking
  - Faceted search

The red bar spans most of the page width, centered horizontally.

ORACLE

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

# Quiz

Which of the following search types are supported by a SearchFormHandler?

- a. Hierarchical
- b. Keyword
- c. File
- d. Text

ORACLE

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

**Answer:** a, b, d

## Summary

In this lesson, you should have learned how to:

- Describe ATG's form-handling capabilities
- Use DSP form tags to attach user input to component properties
- Attach Submit buttons to form handler submit methods
- Use the `RepositoryFormHandler` to create, delete, and update Repository items
- Use a `SearchFormHandler` component to search for Repository items



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

## Additional Information

### *Page Developer's Guide*

- “Forms”
- “Search Forms”
- “Form Handlers”
- “RepositoryFormHandler”
- “DSP Tag Libraries” reference appendix

# Practice 6 Overview: Using Repository Form Handlers

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

This practice covers the following topics:

- Creating forms to add, update, or delete an artist
- Creating a search form handler and form page to search for songs by title or description substring



ORACLE®

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

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

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

## Extending the Profile Repository

ORACLE®

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

# Objectives

After completing this lesson, you should be able to:

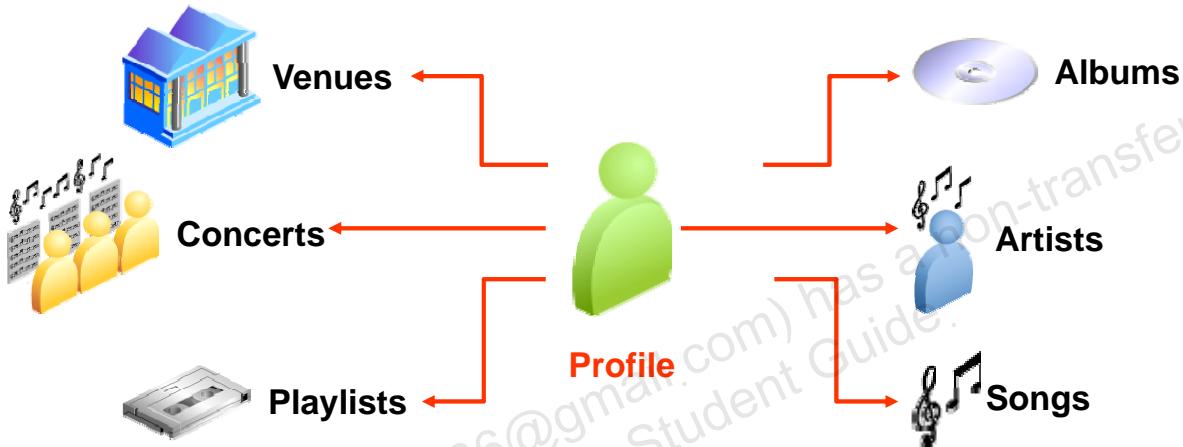
- Explain how the Profile component works
- Create different profile layers
- Modify the Profile Repository by:
  - Adding additional properties
  - Removing unneeded properties
  - Adding new objects and Item Types
- Place properties in categories in the ACC
- Use the ProfileFormHandler component to allow users to log in, log out, register, and update a profile
- Create dynamic web pages that provide ProfileFormHandler functionality



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

## User-Centered Application Design

- Personalized applications are user centered; the Profile should be the central object in your application.
- When considering the user's relationship to other objects always ask, "How can I get there from here?"



ORACLE

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

The user should be the key object in any personalized application.

Relationships between the user and other objects should be established from the user's point of view.

Remember that in almost every stage of the application's execution, the profile is in context.

Think about Dynamusic.com. All the content (concerts, songs, and so on) needs to be accessible by the user. You should have a path in mind for how the user will access the content. In some cases, there may be a direct link, such as a profile attribute: the user may have a list of favorite songs, which is one way of getting from the user to the song items. Sometimes the link is inferred; you know the user's location, so you can do a query to find venues near the user, even though there's no profile attribute for nearby venues. Or, by knowing the user's tastes in music, you can locate artists the user might be interested in. In addition to object relationships, you have to consider how the user will find related content on the site. Will the main page show a list of the user's nearby venues or favorite songs? Or, will the user browse through all venues or songs, and if so, how will he or she find a point to start browsing at. So ask yourself: what types of objects does the user need access to, and how will he or she get there?

## Profile Properties

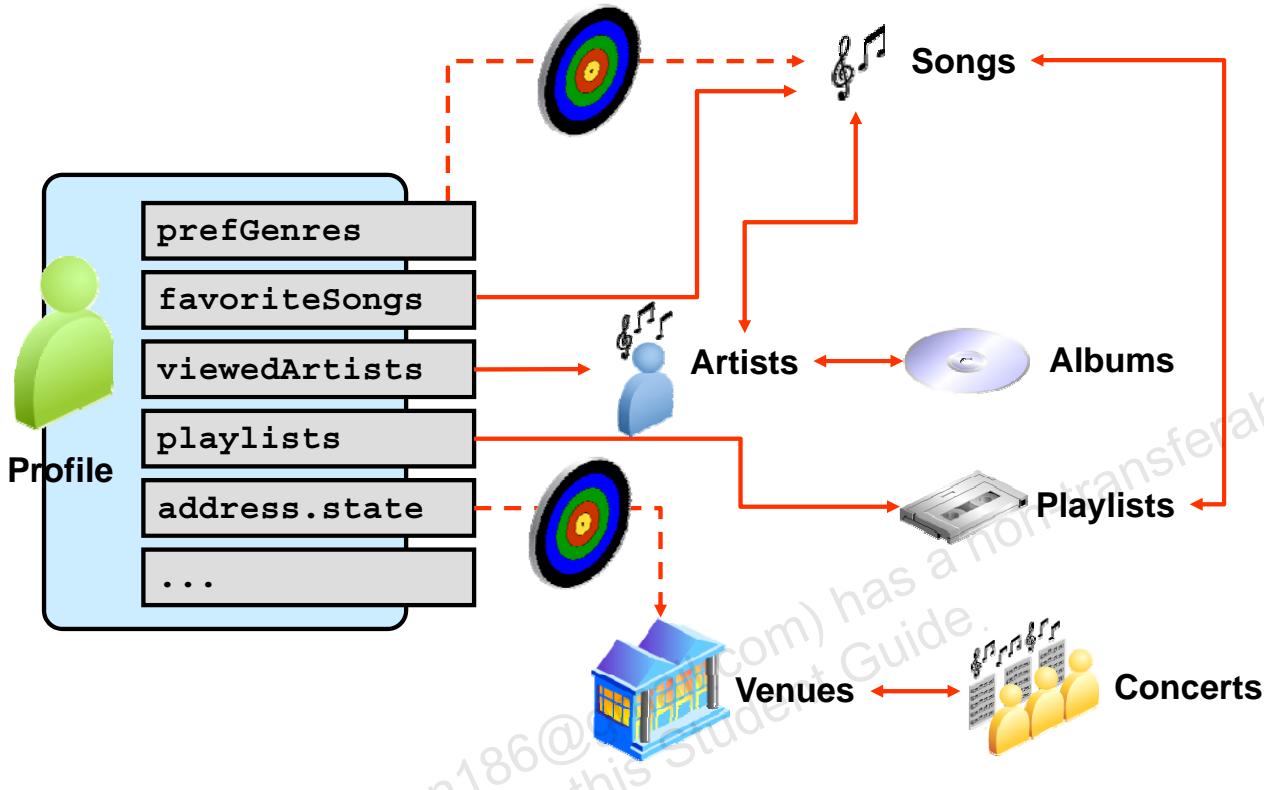
- The default user profile defines many standard user properties.
- Define additional properties that contribute to the definition of a user in the context of the application.
- Add properties to assist in targeting and scenarios.
- Provide properties that link the user to other important business objects.

ORACLE

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

One way of connecting the user to other entities in the application is by adding relevant properties to the Profile.

## Profile Properties: Example



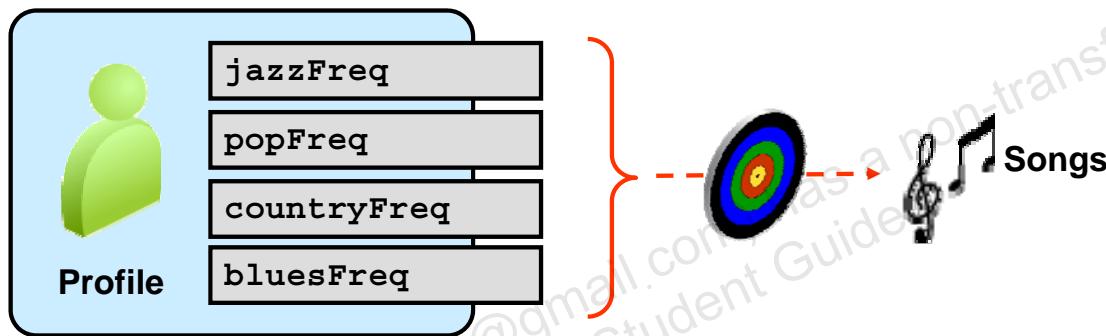
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

In the example in the slide, some profile attributes, such as favoriteSongs and viewedArtists are direct links to associate content, which in turn, provide connections to other objects, such as albums. Some properties, such as prefGenres and state, can be used by targeters to associate the user with relevant content objects.

## Plan for Learning

- Scenarios can detect user actions and change profile attributes, allowing your application to “learn” about users over time.
- Provide profile properties to support this learning.



ORACLE

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

### Example

Infer users' music preferences from the songs they browse and build a scenario to update genre frequency properties in the profile. The relative weight of these frequencies can be used to drive a targeter of suggested song links.

## Plan for Learning: Scenario



ORACLE

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

After you add profile properties to track genre frequency, a scenario can be created to modify those properties whenever a user views an item of the appropriate genre.

## Reasons to Extend the Profile for Dynamusic

- Capture site-specific information
  - What musical genres does the user like?
  - Does the user want to share information with other users?
  - Which items has the user viewed?
- Control content or features
  - Is the user a paid subscriber or guest?



ORACLE

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

## Dynamusic Profile Extensions: Example

- **Preferred Genres:** A set of genres, such as classical or pop
- **Share Profile:** A Boolean to enable users to share their information with other users
- **Viewed Artists:** A list of artist pages the user has viewed
- **Viewed Songs:** A list of song pages the user has viewed
- **Subscriber:** A Boolean to specify if the user is a paid subscriber or guest
- **Info:** Information about the user in the user's own words

ORACLE®

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

Dynamusic needs to consider the following things to extend the profile for its needs:

- Which business objects does Dynamusic need to define to extend the profile?
- How will the new properties be populated?
- How do they want to organize the properties in the ACC?

# Road Map

- Profile repository: Overview
- Profile repository layers
- Profile modifications: Adding a simple property
- Profile modifications: Linking to a different repository item
- Profile form handler



ORACLE

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

## Profile Repository: Overview

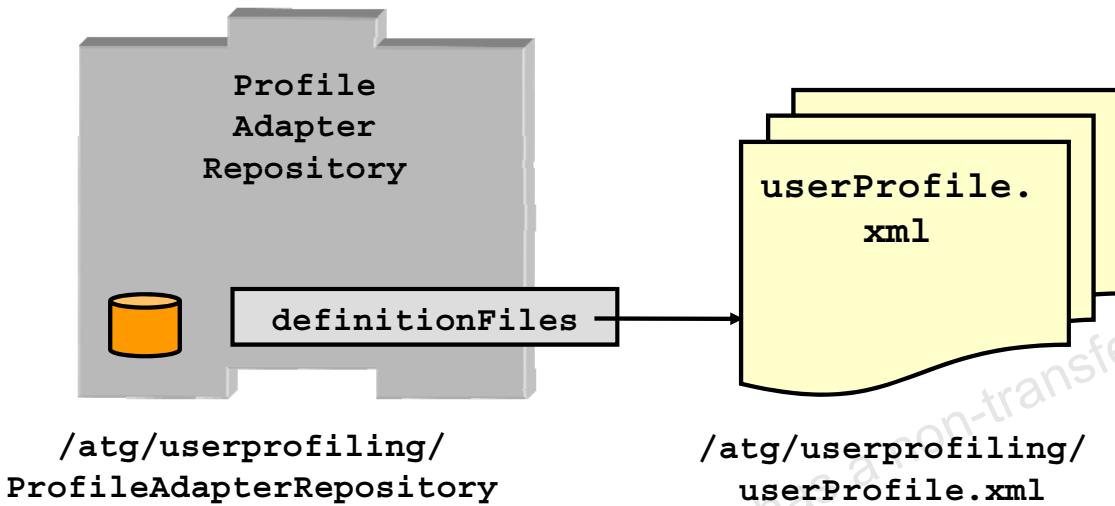
- Every personalized site must manage information about its customers (users).
  - The user data managed by the site is called a profile.
- Out of the box, ATG manages user profiles with a SQL Repository.
- You can also use an LDAP directory.



ORACLE

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

## ProfileAdapterRepository Component



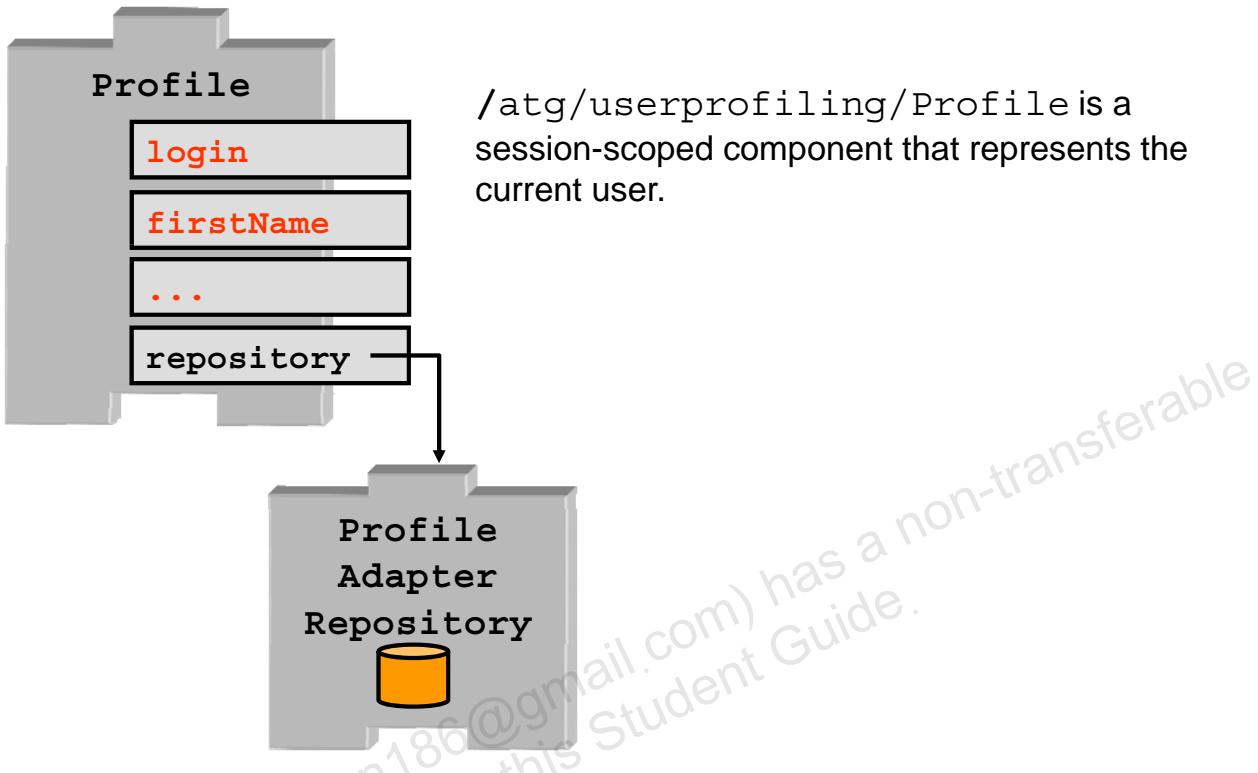
ORACLE

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

The component /atg/userprofiling/ProfileAdapterRepository is a SQL Repository.

Its definitionFiles property points to the <CONFIG>/atg/userprofiling/userProfile.xml file.

## Profile Component



ORACLE

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

In the diagram in the slide, the **Profile** component is a *dynamic bean* and properties such as **login** and **firstName** are *dynamic properties*. A dynamic property is a component property that is pulled from a repository rather than coded into the component with get and set methods.

The **Profile** component represents the current user. It is a central component for all ATG products. It serves as a dynamic bean exposing all of the properties from the `userProfile.xml` file.

The property named “repository” in the diagram in the slide is an expert property. This means that it is hidden by default when viewing the component in the ACC. To view expert properties, go to Tools > View Expert Properties.

You can view individual profile information through the ATG Administration UI by clicking the Personalization Administration link and then the “Search and Edit a Visitor Profile” link.

## Internal and External Users

- External users:
  - Site's registered customers or viewers
  - Stored in `ProfileAdapterRepository`
- Internal users:
  - Users who work for the organization and need access to internal-facing applications (for example, BCC and CSC)
  - Stored in `InternalProfileRepository`
- Both repositories can be extended as needed.
  - This lesson focuses on external users.



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

The full path to the internal Profile Repository is  
`/atg/userprofiling/InternalProfileRepository`.

Its definition file is `/atg/userprofiling/internalUserProfile.xml`.

# Road Map

- Profile repository: Overview
- **Profile repository layers**
- Profile modifications: Adding a simple property
- Profile modifications: Linking to a different repository item
- Profile form handler

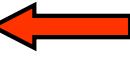


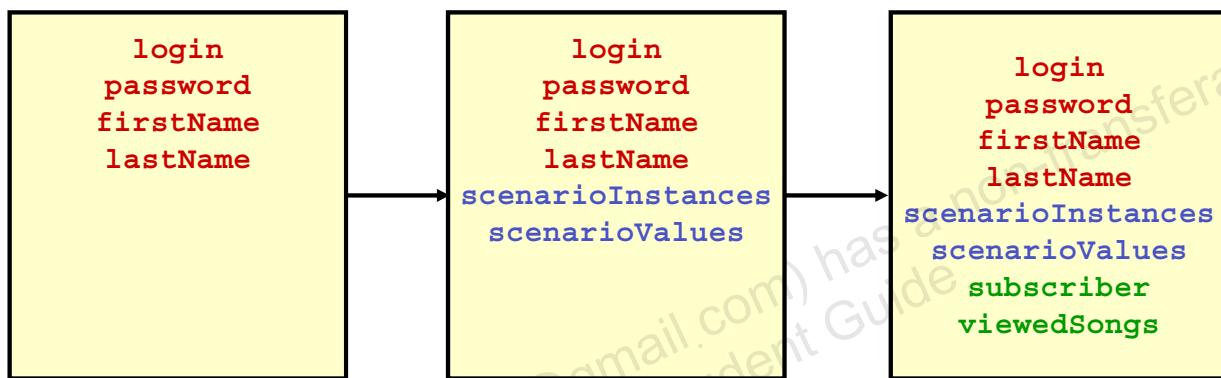
ORACLE

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

# Profile Repository Layers

CONFIGPATH includes:

/DPS/config/profile.jar/atg/userprofiling/userProfile.xml  
/DSS/config/config.jar/atg/userprofiling/userProfile.xml  
**/Dynamusic/config/atg/userprofiling/userProfile.xml** 



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

The Profile Repository uses ATG's XML file combination to construct the definition file. By default, the ProfileAdapterRepository component has its definitionFiles property set to /atg/userprofiling/userProfile.xml.

When the repository starts up, it searches for the XML file in each layer of the CONFIGPATH.

Each of the ATG application modules in the slide has its own CONFIGPATH layer and each one modifies the userProfile.xml file.

The final Repository Definition file can also be affected by the site's localconfig or other layers in the CONFIGPATH.

To view the individual XML files included with the ATG Scenario Engine, you will need to "unjar" the appropriate JAR file (for example, <atgdir>/DPS/config/profile.jar). The WinZip utility works well for viewing the contents of a .jar file.

To view the *combined* XML, use the ATG Admin UI (discussed later in this lesson).

## Base Layer Profile Repository Items

The base ATG platform profile repository defines several Item Types, such as:

- **user:** Defines general user properties (for example, login, firstName, and so on)
- **contact-info:** Defines properties for addresses (for example, city, state, and so on)
- **Mailing:** Defines properties for email campaigns (for example, from, subject, replyTo, and so on)



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

## Modifying the Profile Repository

The Profile Repository can be modified by:

- Adding additional properties
- Removing unneeded properties
- Adding new business objects and Item Types



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

## Extending the Profile Repository

- “Extending” a repository refers to adding properties and Item Types to (or removing from) the base definition.
- Extending the Profile Repository involves two basic steps:
  1. Modify the database schema.
  2. Create a new `userProfile.xml` file layer.



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

Modules can build upon dependent modules' repository definitions, adding, modifying, or even removing items and properties defined in those modules. This is done by “layering” the XML Repository Definition file using ATG’s XML file combination rules to make changes to the repository definition.

## Quiz

Which of the following activities can modify the Profile Repository?

- a. Adding additional properties
- b. Changing the type of a property
- c. Removing unneeded properties
- d. Adding new business objects and Item Types



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

**Answer:** a, c, d

# Road Map

- Profile repository: Overview
- Profile repository layers
- **Profile modifications: Adding a simple property**
- Profile modifications: Linking to a different repository item
- Profile form handler



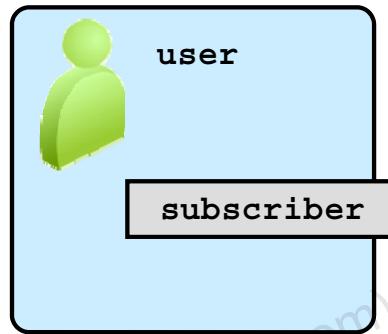
ORACLE

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

## Adding a Simple Property: subscriber

Example: Dynamusic needs to add a Boolean profile property: subscriber.

- Indicates if the user is a paid member

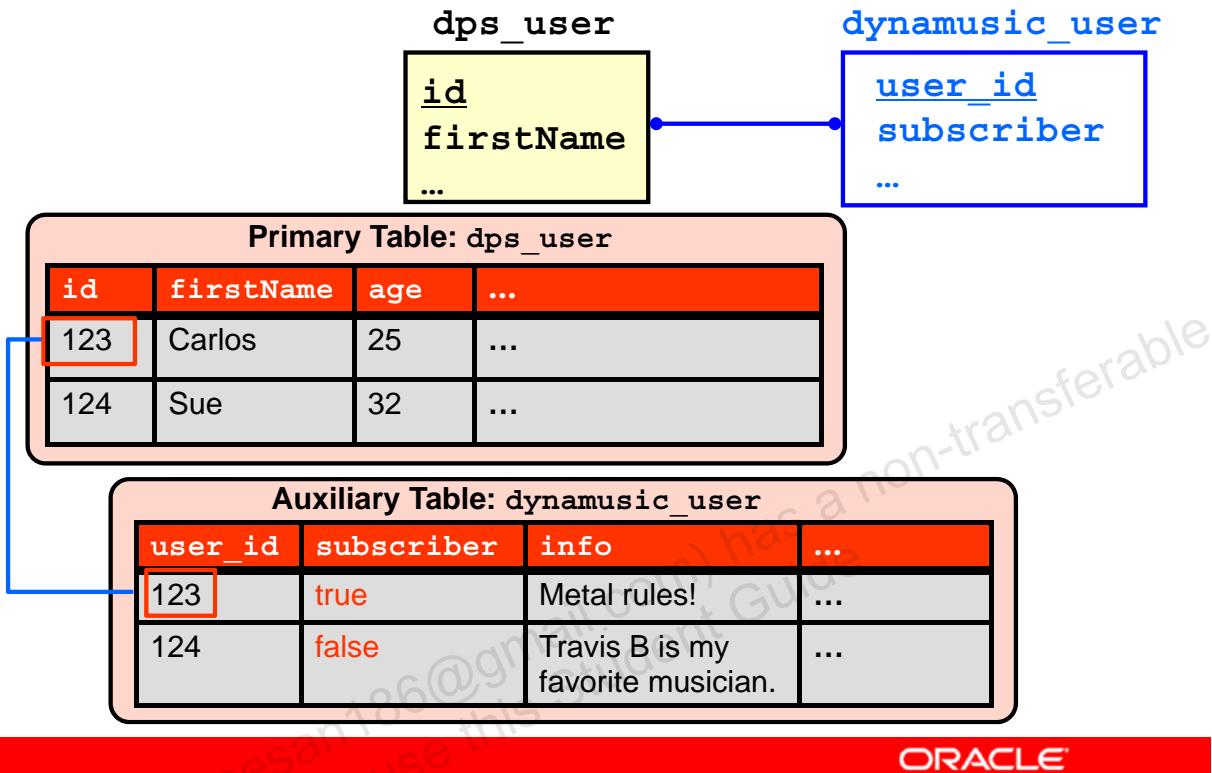


ORACLE®

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

## Step 1: Add an Auxiliary Table

**Best practice:** Add new simple properties to an *auxiliary table*.



ORACLE

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

Every Item Type has one *primary table*. Every item has exactly one row in the primary table. Item types may have any number of *auxiliary tables*. Each item in the primary table may have one row (or zero rows) in an auxiliary table.

Primary tables are joined with auxiliary tables using the repository ID.

When adding new profile properties, you should usually use an auxiliary or multi table, rather than just adding it to the default `dps_user` table. You could argue that adding a new column to the `dps_user` table (for simple properties) would remove both the need for an additional table as well as the need for a database join when retrieving the data. Why create additional tables instead?

- Is easier to make changes later. You can always add auxiliary tables without altering the primary table.
- Makes upgrades easier. If you upgrade to a new version of ATG, the `dps_user` table may change. By creating additional tables, you have less work to do to make your application work on the newer version.

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

## Step 2: Create Repository Definition Layer – XML file

Create a new `userProfile.xml` file and place it in your module's `config` folder.

```
<atgdir>/Dynamusic/config/atg/userprofiling/userProfile.xml
```

```
<gsa-template>
    <!-- extend the user item descriptor -->
</gsa-template>
```



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

The `<gsa-template>` tag is used to restrict the scope of the change.

ATG's XML file combination is a generic process and does not have any knowledge of specific tags like those used in the Repository Definition file. Therefore, you must specify all of the tags in which your modification needs to be enclosed.

For example, adding a new column to a table would require you to include the following tags:

```
<gsa-template>
    <item-descriptor name="...">
        <table name="...">
            <!-- your modification goes here -->
        </table>
    </item-descriptor>
</gsa-template>
```

You would replace the "..." above with the name of the `item-descriptor` and `table` that you are modifying.

## Step 2: Create Repository Definition Layer – user Item Descriptor

Add **subscriber** to the **user** Item Descriptor.

.../userProfile.xml

```
<gsa-template>

<!-- extend the user item descriptor --&gt;
&lt;item-descriptor name="user"&gt;
  &lt;table name="dynamusic_user" type="auxiliary"
    id-column-name="user_id"&gt;
    &lt;property name="subscriber" data-type="boolean"
      column-name="subscriber"/&gt;
  &lt;/table&gt;
&lt;/item-descriptor&gt;

&lt;/gsa-template&gt;</pre>
```

Auxiliary Table: dynamusic_user		
user_id	subscriber	...
123	true	...
124	false	...

ORACLE

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

The example in the slide adds a new table to the definition of a “user.” The `dps_user` table is still being used as the primary table. Because this is an auxiliary table being added, the `id-column-name` is needed so that the system knows how to create the database join with the primary table.

Recall that the `user` Item Descriptor has already been defined in a prior XML file. The new properties will be appended as part of the combined XML file.

In the example in the slide, you add a simple Boolean property to the user profile.

## Road Map

- Profile repository: Overview
- Profile repository layers
- Profile modifications: Adding a simple property
- Profile modifications: Linking to a different repository item
- Profile form handler



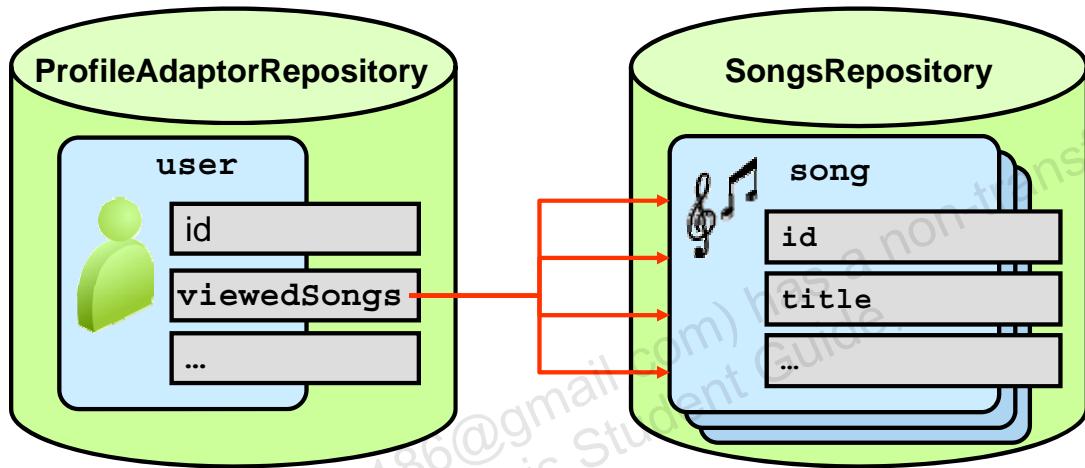
ORACLE

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

## Example: Add a Business Object to the Profile

Dynamusic needs to add a profile property to point to another business object.

- viewedSongs: A set of pointers to all songs each user has viewed

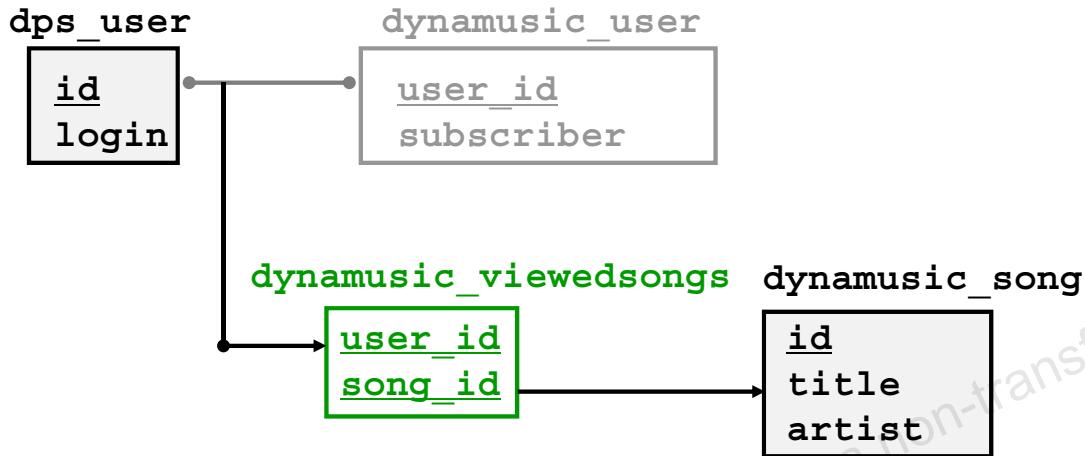


ORACLE

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

In the example in the slide, a link is being added between the profile and an object in another repository, but it is also possible to add new business objects to the same repository.

## Step 1: Add Necessary Tables



**ORACLE**

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

The example in this slide expands on the database modeling slide you saw earlier. In this case, two tables are involved with the `viewedSongs` property: one new multi table to map the user ID to the song ID and the other a primary table that holds the song data. The table to hold the song data was provided in the lab setup as part of the SongsRepository, configured in the Dynamusic-Base application module.

The SQL to create the table for this example is:

```

CREATE TABLE dynamusic_viewedsongs (
  user_id  VARCHAR(32)  not null references dps_user(id),
  song_id  VARCHAR(32)  not null references song(id),
  primary key(user_id, song_id)
);
  
```

## Step 2: Add viewedSongs to the user Item

userProfile.xml

```
...
<item-descriptor name="user">
  <table name="dynamusic_user" type="auxiliary"
    id-column-name="user_id">
    <property name="subscriber" data-type="boolean"
      column-name="subscriber"/>
  </table>
  <table name="dynamusic_viewedsongs" type="multi"
    id-column-name="user_id">
    <property name="viewedSongs"
      data-type="set" component-item-type="song"
      repository="/dynamusic/SongsRepository"
      column-name="song_id" />
  </table>
</item-descriptor>
...

```

ORACLE

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

The example in the slide is a set of item pointers and is referring to a component-item-type of song. In this example, the song Item Type is not defined within the same repository. Therefore, you need to define the repository attribute with the Nucleus name of the repository component.

## Removing Properties via Layering

Repository definition layers can remove properties as well as add them:

userProfile.xml

```
<gsa-template>
  <item-descriptor name="user">
    <table name="dps_user">
      <property name="gender"
        xml-combine="remove"/>
    </table>
  </item-descriptor>
</gsa-template>
```



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

For more details about removing properties, refer to the XML file combination rules as detailed in the “Nucleus: Organizing JavaBean Components” chapter of the *Programming Guide*.

## Transient Properties

- Most properties mirror values in database tables.
- Some of the properties are termed *transient*. They are:
  - Not stored in the database
  - Declared outside all `<table>` tags
- Example: `lastItemViewed`

```
<item-descriptor name="user">
  <property name="lastItemViewed"
    data-type="string"
    display-name="Last Item Viewed"/>
  </property>
  ...
</item-descriptor>
```



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

Transient properties are not specific to the Profile Repository. They are, however, often very useful in the context of the profile for temporarily storing information during the user's session. In the example in the slide, Dynamusic would like to track which item the user last viewed to display on a subsequent page as a link back to the item. This information is important only within a single session, and therefore, does not need to be stored in the database.

## Viewing Combined Layers

ATG's Admin UI provides an option to view the combined XML file.

1. Access the admin interface:  
`http://localhost:7103/dyn/admin`
2. Use the Component Browser to browse to the Nucleus component:  
`/atg/userprofiling/ProfileAdapterRepository`
3. Click the `definitionFiles` property.



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

A helpful debugging tip is to view the combined XML file. This shows you the final file (built using all the config layers) that ATG uses to construct the Item Descriptors for this repository. This technique is not specific to the Profile Repository; it works for any SQL Repository.

## Instructor Demonstration: Combined XML File

CONFIGPATH	/atg/userprofiling/userProfile.xml
filename	
Source files	<ul style="list-style-type: none"> <li>• C:\AE10\ATG10.0.1\DPS\config\profile.jar\atg\userprofiling\userProfile.xml</li> <li>• C:\AE10\ATG10.0.1\DS\config.jar\atg\userprofiling\userProfile.xml</li> <li>• C:\AE10\ATG10.0.1\Dynamicusic\config\atg\userprofiling\userProfile.xml</li> </ul>
System Id (DTD name)	http://www.atg.com/dtds/gsa/gsa_1.0.dtd
XML value	<pre>&lt;?xml version="1.0" encoding="UTF-8"?&gt;  &lt;!DOCTYPE gsa-template SYSTEM "dynamicsystemresource:/atg/dtds/gsa/gsa_1.0.dtd"&gt;  &lt;!-- This file provides a starting point for extending the      profile repository.       The comments below coincide with the instructions in the      lab. --&amp;gt; &amp;lt;gsa-template&amp;gt;  &amp;lt;header&amp;gt;     &amp;lt;name&amp;gt;DPS User Profile&amp;lt;/name&amp;gt;     &amp;lt;author&amp;gt;Jeff&amp;lt;/author&amp;gt;     &amp;lt;version&amp;gt;\$Id: //product/DPS/version/10.0.1/templates/DPS/config/profile/atg/u &amp;lt;/header&amp;gt;  &amp;lt;item-descriptor expert="false" display-name-resource="itemDescriptorUser" id-s &amp;lt;attribute name="resourceBundle" value="atg.userprofiling.UserProfileTemplate" &amp;lt;attribute name="icon" value="userIcon"/&amp;gt;  &amp;lt;attribute name="categoryBasicsPriority" value="10"/&amp;gt; &amp;lt;attribute name="categoryEmailPriority" value="20"/&amp;gt; &amp;lt;attribute name="categoryActivityPriority" value="30"/&amp;gt;&lt;/pre&gt; </pre>



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

This page is <http://localhost:7103/dyn/admin/nucleus/atg/userprofiling/ProfileAdapterRepository?propertyName=definitionFiles>, which can be viewed by going to the Administrative Interface (<http://localhost:7103/dyn/admin>), going to the Component Browser, browsing to the Repository component, and selecting the definitionFiles property.

## Property Attributes

```
<property name="subscriber" data-type="boolean"
column-name="subscriber" category="dynamusic"
default="false" display-name="Member"/>
```

User "Donna Whynot"		Profile	Roles
Registration date	April 15, 1999 12:00:00 AM EDT		
Business Process Marker Info			
This profile's set of business process markers			
dynamusic			
info	I'd like to see The Purpose Spit live. Anyone...		
Member	false		
prefGenres	[pop, country]		
shareProfile	true		
viewedArtists	[Travis B., The Porpoise Spit]		
viewedSongs	[Any Man Can See, Bullets Over The City, Ca...		
Marker Info			
This profile's list of markers			



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

The category attribute helps organize profile properties in both the ACC and the Business Control Center (BCC). If it is not specified, the property will show in the “Uncategorized” category. The default attribute is used to specify the property’s default value. In the example in the slide, subscriber should be false unless the user has a paid subscription. The display-name attribute controls the text that appears in the ACC for the property name. The following are the other attributes that can change the ACC presentation:

- required: Is false by default. Required fields are shown prefixed with an asterisk.
- hidden: Hides the property in the ACC, but can be accessed programmatically; might be used for calculated data or for data that you would like to hide from ACC users (for example, credit card number)
- enumerated (data-type): Enumerated data types provide a drop-down list of the enumerated options
- property-type: Specifies a Java class that can return a “dynamic” property value
- item-type: Specifies that the value of an item is the ID of another item-descriptor; the ACC will display the icon and a lookup dialog box will be presented to select a value (for example, the Profile’s home address property)
- writable=false: Shows up as light gray and cannot be modified by the ACC user
- expert: Appears only to users who have the expert check box selected as an ACC option

# Road Map

- Profile repository: Overview
- Profile repository layers
- Profile modifications: Adding a simple property
- Profile modifications: Linking to a different repository item
- Profile form handler



ORACLE

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

## Dynamusic Profile Forms

- Dynamusic needs to create pages for users to create and manage their user profiles.

**Update Profile**

---

Login: ann

First name:

Last name:

State:

- Use `ProfileFormHandler` for this functionality.

ORACLE®

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

## ProfileFormHandler

- Nucleus component that facilitates the creation of the following types of pages:
  - Login
  - Logout
  - Registration
  - Update
  - Password Change
- Similar to RepositoryFormHandler



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

ProfileFormHandler is a repository form handler specific to the Profile Repository. It is an instance of the `atg.scenario.userprofiling.ScenarioProfileFormHandler` class. This class and the `atg.repository.servlet.RepositoryFormHandler` class are both subclasses of `atg.droplet.GenericFormHandler`.

**Note:** The ACC has a wizard to help with the creation of profile pages for operations such as login and registration. For more information, refer to the *Page Developer's Guide*, Appendix E: ATG Document Editor.

## Accessing Profile Attributes in a Form

newProfile.jsp

```
<dsp:form action="newProfile.jsp" method="post">  
...  
Login Name:  
<dsp:input type="text"  
bean="ProfileFormHandler.value.firstName"/>  
...  
Birthday (MM/DD/YYYY) ::  
<dsp:input type="text"  
bean="ProfileFormHandler.value.dateOfBirth"  
date="M/dd/yyyy" maxlength="10" size="10"/>  
...  
</dsp:form>
```

* First name:	Jennifer
* Last name:	Davis
Birthday (MM/DD/YYYY):	12/21/1980
State:	MA

ORACLE

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

As you learned with RepositoryFormHandler, ProfileFormHandler uses a value Dictionary to temporarily store the attribute. It can then perform error checking before writing new information to the data source. This functionality is available through dynamic beans.

## Profile Submission Handler Methods

Submission properties of the ProfileFormHandler include:

- **create**: Registers a new user
- **update**: Updates an existing user's profile
- **login**: Verifies login and password; sets the current profile to that of user
- **logout**: Resets the current profile to anonymous user
- **changePassword**: Changes the current user's password

Example:

```
<dsp:input type="submit" value="Login"  
bean="ProfileFormHandler.login"/>
```



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

You may recall that RepositoryFormHandler has the following submission properties as discussed in the lesson titled “Using Repository Form Handlers”:

- handleCreate()
- handleUpdate()
- handleCancel()
- handleDelete()

These are also available to the ProfileFormHandler, as well as additional handle methods:

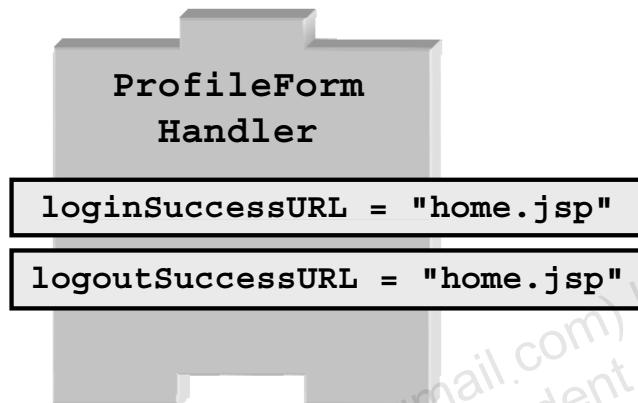
- handleLogin()
- handleLogout()
- handleChangePassword()

ATG provides a handy component for creating and enforcing password rules. The /atg/userprofiling/passwordchecker/PasswordRuleChecker component has an enabled property. When set to true, the component looks at the rules property for the list of components that contain the rules, such as PasswordMinLengthRule and PasswordMustIncludeSymbolRule. You can easily view this component in the ACC.

## Profile Page Redirection

ProfileFormHandler has both success and error redirection properties for each submit property, such as loginSuccessURL and loginErrorURL.

Example:



ORACLE

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

Additional redirection properties include logoutSuccessURL, logoutErrorURL, changePasswordSuccessURL, and so on. If any of these properties are not set, and if the associated condition occurs, the form's "action" page will be displayed.

## What the login Handler Does

Profile represents an anonymous user until login. A session has already been created for the user. Upon submit, the login process starts:

1. Validate user/password.
2. Load values from the Profile Repository.
3. Optionally and selectively, copy anonymous properties from the current session to the Profile component.
4. Set the dataSource property of the Profile component.



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

The Profile component is instantiated by the ProfileRequestServlet. Until the user logs in, the user is anonymous. However, because the Profile component is used by other ATG technologies (such as Targeting), it can be used to gather characteristics about the guest. The session can still be personalized implicitly.

If you want to save the values of session properties into the Profile component, you can optionally set the ProfileFormHandler.propertiesToCopyOnLogin property to the session properties you selectively want to persist.

The dataSource property points to the actual repository item from the Profile Repository.

## ProfileFormHandler Properties

Additional properties to customize profile form handling:

- boolean **confirmPassword**
- boolean **checkForRequiredProperties**
- boolean **extractDefaultValuesFromProfile**



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

- confirmPassword: If true, the form verifies that the user entered the same thing for value.password and value.confirmPassword.
- checkForRequiredProperties: If true, the form will raise an error if the user does not enter any of the attributes marked “required.”
- extractDefaultValuesFromProfile: If true, the form will initially display the contents of the current user profile. This is typically set to true for update pages and set to false for new user forms (registration pages). The default value is true.

## Quiz

The ProfileFormHandler allows you to do the following to a user:

- a. Log out
- b. Log in
- c. Delete
- d. Update the profile
- e. Register



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

**Answer: a, b, d, e**

## Summary

In this lesson, you should have learned how to:

- Extend ATG-supplied repositories (such as the Profile Repository) by using XML file combination
- Use the `ProfileFormHandler` to handle forms for registration, login, logout, update, and password selection



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

## For More Information

- *Programming Guide*
  - “Nucleus: Organizing JavaBean Components”
    - Subsection “XML File Combination”
- *Repository Guide*
  - “SQL Repository Reference”
- *Page Developer’s Guide*
  - “User Profile Forms”



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

# Practice 7 Overview: Extending the Profile Repository

This practice covers the following topics:

- Adding new properties to the profile:
  - info (simple String property)
  - shareProfile (simple Boolean property)
  - prefGenres (multi String property)
  - viewedArtists (multitem property)
- Creating a registration page



ORACLE

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

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

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