ATG Repositories Advanced Topics

Presenter's Name
Presenter's Title

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Oracle Training Materials – Usage Agreement

Use of this Site ("Site") or Materials constitutes agreement with the following terms and conditions:

1. Oracle Corporation ("Oracle") is pleased to allow its business partner ("Partner") to download and copy the information, documents, and the online training courses (collectively, "Materials") found on this Site. The use of the Materials is restricted to the non-commercial, internal training of the Partner's employees only. The Materials may not be used for training, promotion, or sales to customers or other partners or third parties.

2. All the Materials are trademarks of Oracle and are proprietary information of Oracle. Partner or other third party at no time has any right to resell, redistribute or create derivative works from the Materials.

3. Oracle disclaims any warranties or representations as to the accuracy or completeness of any Materials. Materials are provided "as is" without warranty of any kind, either express or implied, including without limitation warranties of merchantability, fitness for a particular purpose, and non-infringement.

4. Under no circumstances shall Oracle or the Oracle Authorized Delivery Partner be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this Site of Materials. As a condition of use of the Materials, Partner agrees to indemnify Oracle from and against any and all actions, claims, losses, damages, liabilities and expenses (including reasonable attorneys' fees) arising out of Partner's use of the Materials.

5. Reference materials including but not limited to those identified in the Boot Camp manifest can not be redistributed in any format without Oracle written consent.

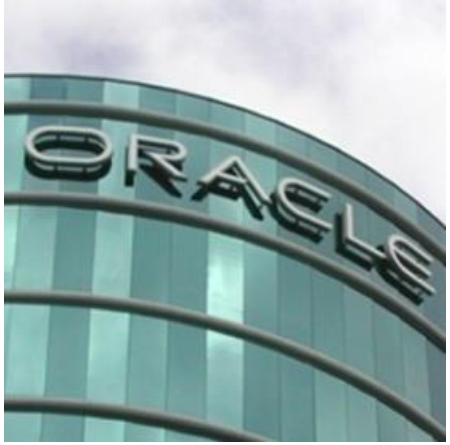
Agenda

- Derived and user defined properties
- Repository API

Learning Objectives

At the end of this lesson you should be able to:

- Create user-defined properties to custom functionality to repositories
- Use derived properties to derive the property value from other repository properties
- Learn about ATG Repository API
- Understand the core repository API elements
- Use query builder to programatically query the repository



Section 1:
User Defined Properties and Derived Properties

User-Defined Property Types

- In addition to the standard data types, the SQL Repository lets you add your own data types of properties.
- It uses custom Java classes that provide special handling for repository item properties.
- Examples:
  - Transient properties computed "on the fly" from values from the database. Compute age of a user given the birthday.
  - Values which must be modified or processed before storage or after retrieval from the database.

Creating a Custom Property Type

- Create a property descriptor java class which extends:
  - **`GSAPropertyDescriptor: stored property`**
  - **`RepositoryPropertyDescriptor: transient property`**
- Set property-type to your created class in the XML item descriptor property tag.

Property Descriptor Objects

- A Property Descriptor has methods to handle how property values are stored or retrieved:
  - `getPropertyValue()`
  - `setPropertyValue()`
- Developers may override any of the methods to change how the property attributes are handled.
- A Property Descriptor has methods to set all the attributes of a property.

Example: Age Calculator - Class

- Calculate the age of user given the birthday.

```
Class AgeCalculator extend GSAPropertyDescriptor{
   ...
   public Object getPropertyValue(RepositoryItemImpl pItem,
                                  Object pValue) {
     int age=0;
     Date birthDay = (Date) pItem.getPropertyValue("birthDay");

     Calendar dob = Calendar.getInstance();
     dob.setTime(birthDay);
     Calendar today = Calendar.getInstance();
     age = today.get(Calendar.YEAR) - dob.get(Calendar.YEAR);

     return Integer.valueOf(age);
  }
  ...
}
```

Example: Age Calculator – Item Descriptor

- Use the property-type attribute to specify your implementation class.

```
 <item-descriptor name="user">
   <property name="loggedIn" data-type="boolean">
   <table name="user" type="primary" id-column-names="id">
     <property name="login" data-type="string"
               column-name="login">
     <property name="birthDay" data-type="date"
               column-name="birth_day">
   </table>
   <property name="age" writable="false" data-type="int"
        property-type="com.myproject.AgeCalculator" />
</item-descriptor>
```

Property Attributes

- User-defined properties may use custom attributes to control behavior.
- Those attributes are passed to the setValue method of property descriptor.

```
<item-descriptor name="user">
  <property name="loggedIn" data-type="boolean">
  <table name="user" type="primary" id-column-names="id">
    <property name="login" data-type="string"
              column-name="login">
    <property name="birthDay" data-type="date"
              column-name="birth_day">
  </table>
  <property name="age" writable="false" data-type="string"
       property-type="com.myproject.AgeCalculator">
     <attribute name="format" value="inWords"/>
  </property>
</item-descriptor>
```

Example: Property Attributes – Class SetValue Method

- The SetValue is called for each attribute. Capture it and store the format.

```
// AgeCalculator.java
. . .
String format = "inNumbers";
public void setValue(String pAttributeName, Object pValue) {
  super.setValue(pAttributeName, pValue);

  if (pAttributeName.equalsIgnoreCase("format")){
    String lFormat = pValue.toString();
    if (lFormat.equalsIgnoreCase("inWords"))
      this.format = "inWords";
       else if (lFormat.equalsIgnoreCase("inNumbers"))
      this.format = "inNumbers";
    else // Should log error
      this.format="inNumbers";
  }
}
```

Example: Property Attributes – getPropertyValue Method

```
Class AgeCalculator extend GSAPropertyDescriptor{
   ...
   public Object getPropertyValue(RepositoryItemImpl pItem,
                                  Object pValue) {
     int age=0;
     Date birthDay = (Date) pItem.getPropertyValue("birthDay");

     Calendar dob = Calendar.getInstance();
     dob.setTime(birthDay);
     Calendar today = Calendar.getInstance();
     age = today.get(Calendar.YEAR) - dob.get(Calendar.YEAR);
     if (format.equals("inWords"))
        return convertToWords(age);
     else
        return String.valueOf(age);
   }
   ...
}
```

Derived Properties

- Derived properties derive their property values from another repository item or from another property.
- Derived properties must be transient properties.
- They are important to data structures that use a tree structure.
- Examples can be:
  - Can search through a series of properties to find a non-null value.
  - Employee has spending limits. If it is not set, the value of employee's department spending limit property is used.

Derivation Syntax

- Derived properties are defined through derivation tag.
- Each derivation tag encloses one or more expression tag(s).
  - Each expression tag encloses a property name.
  - Both the derived property and the expressions must be of the same type.
- By default, the repository will return the value of the first property in the list that is not null.

```
<property name="propertyName">
    <derivation>
        <expression> property1 </expression>
        <expression> property2 </expression>
        ...
    </derivation>
</property>
```

Example : Derived Properties (1)

- The following derivation uses the firstNonNull method to compute the spending limit.
- The result is stored in the property spendingLimit.

```
<item-descriptor name="employee">
   ...
   <property name="department" item-type="department"/>
   <property name="empSpendingLimit" data-type="int"/>
   <property name="spendingLimit" writable="false">
      <derivation method="firstNonNull">
         <expression>empSpendingLimit</expression>
         <expression>department.deptSpendingLimit</expression>
      </derivation>
</property>
</item-descriptor>

<item-descriptor name="department">
   ...
   <property name="deptSpendingLimit" data-type="int"/>
</item-descriptor>
```

Example : Derived Properties (2)

- In the previous example:
  - Employee's spendingLimit property is a derived property.
  - It will check first expression empSpendingLimit. If empSpendingLimit is not null, the value of empSpendingLimit will be used .
  - If empSpendingLimit is null, it will check the second expression department.deptSpendingLimit (its departement's deptSpendingLimit).
  - If department.deptSpendingLimit is not null, the value of department.deptSpendingLimit will be used.
  - Otherwise spendingLimit property value is null.

Derivation Methods

- Attribute method of derivation tag can specify one of several different derivation methods:
  - firstNonNull,
  - firstWithAttribute,
  - firstWithLocale,
  - alias,
  - union,
  - collectiveUnion.
- Method firstNonNull is default setting.
- User can define custom derivation method.

Derivation Method: firstWithLocale

- Performs the following actions:
  - Gets the user's current locale as the key.
  - Compares this locale to each expression's locale value.
  - Returns the first property matched.
- The properties are searched in a locale-specific way.
  - For example, if current locale is fr_FR_EURO, then the search order is : fr_FR_EURO, fr_FR, fr. Then first matched property is returned.
- Can set attribute defaultKey which is used when the current local key is null.

Example: Derivation Method - firstWithLocale

- If the current user local is de, then property value of displayName_de is returned.

```
<property name="displayName data-type="string">
   <derivation method="firstWithLocale">
     <expression>displayName_en</expression>
     <expression>displayName_de</expression>
   </derivation>
   <attribute name="defaultKey" value="en"/>
   <attribute name="derivationAttribute" value="locale"/>
   <attribute name="keyService"
           value="/atg/userprofiling/LocaleService"/>
   <attribute name="keySubProperty" value="locale"/>
</property>
```

Section 1 
Check Your Understanding

What are user defined properties?

Answer: They allow you to define a property on a repository item and provide a class implementation used to respond to it when the property is accessed.

Section 1 ✔
Check Your Understanding

How can you pass additional values to the user defined property descriptor to control its behavior?

Answer: You can use the attribute tag to pass additional values that can be used to control the behavior of the property.

Section 1 ✔
Check Your Understanding

What are derived properties?

Answer: Derived properties derive their value from other properties based on criterion such as non null, locale etc.

Section 1
Check Your Understanding

List common derivation methods for derived properties.

Answer: firstNonNull, firstWithLocale, alias, union etc.

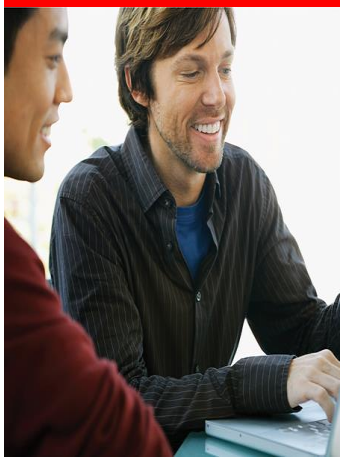
Section 1 ✔
Check Your Understanding

What tags are used to configure derived properties in the item descriptor?

Answer: Derivation and expression tags are used to specify derived properties.

Summary

- User defined properties and derived properties are ways to extend the concept of repositories to handle custom situations.
- User defined properties let you add your own properties to repository items and have a java class respond with results when the property is requested.
- Derived Properties derive their property value from another repository item or from another property.
- firstNonNull, firstWithAttribute, union are some of the supported derivation methods.



Section 3:
Repository API

Repository API Overview

- The ATG Repository API (atg.repository.*) is the foundation of persistent object storage, user profiling, and content targeting in ATG products.
- A repository is a data access layer that defines a generic representation of a data store.
- The Repository API is the generic representation that developers use to access the underlying data.

Core Repository API Elements

- atg.repository.Repository
  - An interface that provides methods to access RepositoryItems, RepositoryViews, and ItemDescriptors.
- atg.repository.RepositoryView
  - Used to search for items in the repository if you do not have an exact repository ID.
- atg.repository.RepositoryItem
  - An immutable interface that represents the elements in a repository.
- atg.repository.MutableRepository
  - Adds services to create, add, remove, and update items.
- atg.repository.PropertiesChangedEvent
  - The event that is broadcast when a repository item is modified.

Mutable Repository

- Base interfaces of a repository define an immutable data store.
- Some repository services implement MutableRepository.
- The SQL repository implements this interface.
- The MutableRepository interface defines these methods:
  - createItem(),
  - addItem(),
  - removeItem(),
  - updateItem().

Retrieving a Repository Item by ID

- **Use the following methods of the Repository object to retrieve a repository item by ID.**

```
RepositoryItem getItem(String pId, String pDescriptorName)

RepositoryItem[] getItems(String[] pIds,
                          String pDescriptorName)

RepositoryItem getItem(CompositeKey pId,
                          String pDescriptorName)

RepositoryItem [] getItems(CompositeKey [] pIds,
                               String pDescriptorName)
```

- **Examples:**

```
RepositoryItem user =
       myRepository.getItem("user1003","user");

RepositroyItem [] users = myRepository.getItem(
      String[]{"user1001","user10002"},"user")
```

Create a New Item

- There are two createItem methods:
  - createItem(String pDescriptorName),
  - createItem(String pId, String pDescriptorName).
- The createItem methods return a transient instance of a MutableRepositoryItem.

```
MutableRepositoryItem item =
        myRepository.createItem("user")

MutableRepositoryItem item =
        myRepository.createItem("user1003","user")
```

Add a New Item

- After you create an item, you can turn it into a persistent repository item with the addItem method.
- Call addItem methods:

```
try{
  MutbleRepositoryItem item =
            myRepository.createItem("user")
  RepositoryItem user = myRepository.addItem(item);
} catch(RepositoryException e){
 // log the exception
}
```

Update an Item

- Fetch a mutable version of the repository item through the getItemForUpdate and getItemsForUpdate methods.
- Use the setPropertyValue method of MutableRepositoryItem to change as many properties as you wish.
- Save the changes with the updateItem method.
- Depending on the transactional context, the update can be committed immediately or it can happen when the associated transaction commits.

Update Item Example

```
try {
  RepositoryItem user = ... // get a reference to the user
  MutableRepository mutableRepository
                   =(MutableRepository)user.getRepository();
  MutableRepositoryItem mutableUser =
       mutableRepository.getItemForUpdate(user.getRepositoryId(),
              user.getItemDescriptor().getItemDescriptorName());

  mutableUser.setPropertyValue("name", "bob");
  mutableUser.setPropertyValue("age", new Integer(26));
  mutableRepository.updateItem(mutableUser);
}
catch (RepositoryException exc) {
  // Log the exception
}
```

Remove Item

- Call remove method and pass the ID and ItemDescriptor name of the item that needs to be removed.

```
try{
  MutableRepository mutableRepository =
        (MutableRepository)getRepository();
  mutableRepository.removeItem(userid, "user101");
} catch(RepositoryException e){
   // Log the exception
}
```

- The item's property values are deleted and are no longer accessible from the repository.

Querying the Repository

- A repository query defines a request to find all items of a specified item type that fit a set of criteria.
- The Repository API can express a wide variety of queries:
  - queries that match patterns in text,
  - query through collections,
  - query through complex values.
- Queries can be built and executed with the Repository API or can be represented in the Repository Query Language (RQL).

Parameters in Queries

- ?{number} represents a parameterized value that is supplied on query execution.
- Where RQL queries are used to represent finder methods, the parameters are filled in from the arguments of the finder methods.
- Parameter expressions can generally be used wherever constant values are used but cannot be used in array expressions and cannot be used as substitutes for property names.

```
RepositoryView view = repository.getView("person");
RqlStatement statement =
          RqlStatement.parseRqlStatement("age > ?0");
Object params[] = new Object[1];
params[0] = new Integer(23);
RepositoryItem [] items = statement.executeQuery (view,
                                                  params);
```

Repository Query API

- Querying the database can be performed using:
  - RQL Queries,
  - Repository Query API.
- The Repository Query API has two basic elements:
  - Query Builder provided as atg.repository.QueryBuilder,
  - Query Options provided as atg.repository.QueryOptions.
- The Query Builder interface enables you to build Query objects that can be passed to the repository.
- The Query Options class is used to specify ways the query can be modified.

Query Builder

- Query Builder interface defines the available query operations that the repositories support.
- The Query Builder enables you to build the Query Object.
- A Query is constructed from one or more QueryExpressions and a query operation.
- The following logical operations are supported:
  - AND, OR,
  - NOT,
  - EQUALS, GREATER THAN, LESS THAN,
  - GREATER THAN OR EQUALS, LESS THAN OR EQUALS.

Query Builder Steps

- Given a RepositoryView, initialize a QueryBuilder:

```
QueryBuilder qb = view.getQueryBuilder();
```

- Create a QueryExpression for the property:

```
QueryExpression gender =
                  qb.createPropertyQueryExpression("gender");
QueryExpression female =
                  qb.createConstantQueryExpression("female");
```

- Create a comparisonQuery that uses the expressions:

```
Query femaleQuery = qb.createComparisonQuery(
                        gender, female, QueryBuilder.EQUALS);
```

- Pass the resulting Query to the Repository View:

```
items = view.executeQuery(femaleQuery);
```

Query Options

- The QueryOptions class is used to modify the query.
- The QueryOptions class is passed to the executeQuery method or the RepositoryView:

```
RepositoryItem[] executeQuery(
                 Query pQuery, QueryOptions pQueryOptions);
```

- The following are the QueryOptions properties:
  - startingIndex: Index of the first element that should be returned.
  - endingIndex: Index of the last element that should be returned.
  - sortDirectives: Specifies the sort order.
  - precachedPropertyNames: A list of properties to pre cache.

Query Builder Example

```
MutableRepository pRepository =
 (MutableRepository)ServletUtil.getCurrentRequest().resolveName
              ("/atg/userprofiling/ProfileAdapterRepository");

RepositoryItemDescriptor userDesc =
                      pRepository.getItemDescriptor("user");
RepositoryView userView = userDesc.getRepositoryView();
QueryBuilder userBuilder = userView.getQueryBuilder();

QueryExpression userType =
    userBuilder.createPropertyQueryExpression("userType");
QueryExpression two =
    userBuilder.createConstantQueryExpression(new Integer(2));

Query userTypeIsTwo = userBuilder.createComparisonQuery(
                        userType, two, QueryBuilder.EQUALS);

RepositoryItem[] answer = userView.executeQuery(userTypeIsTwo);

System.out.println("running query: userType = 2");
if (answer != null)
  for (int i=0; i<answer.length; i++)
    System.out.println("id: " + answer[i].getRepositoryId());
```

Named Queries

- Developers can define a named query in an item-descriptor element in an SQL repository definition file.
- Use NamedQueryView interface method to create and access named queries.
- Named queries are supported only in SQL and Integration repositories.
- An item descriptor can define named queries in three ways:
  - RQL named queries,
  - SQL named queries,
  - Stored procedures.

Named Queries Example

- UserProfile.XML

```
…
<item-descriptor name="user">
    …
   <named-query>
       <rql-query>
          <query-name>myQuery</query-name>
          <sql>age>20 AND lastName ENDS WITH "son"</rql>
       </rql-query>
   </named-query>
</item-descriptor>
```

- Code

```
RepositoryView userView = getRepository().getView("user");
if(userView instanceof NamedQueryView) {
   NamedQueryView nameView = (NamedQueryView)songView;
   Query namedQuery =
             nameView.getNamedQuery("myQuery");
   RepositoryItem[] items = nameView.executeQuery(namedQuery);
}
```

Section 2 
Check Your Understanding

Name a few core API elements of the Repository API.

Answer: Repository, RepositoryView, RepositoryItem, etc.

Section 2

Check Your Understanding

What interface provides methods to create, add, remove, and update repository items?

Answer: The Mutable repository interface provides these methods.

Section 2 
Check Your Understanding

What are the two methods to query the repository?

Answer: Using RQL and the Query Builder.

Section 2 
Check Your Understanding

What are the main classes used in Query Builder?

Answer: QueryBuilder, QueryOptions, QueryExpression, and Query.

Section 2 
Check Your Understanding

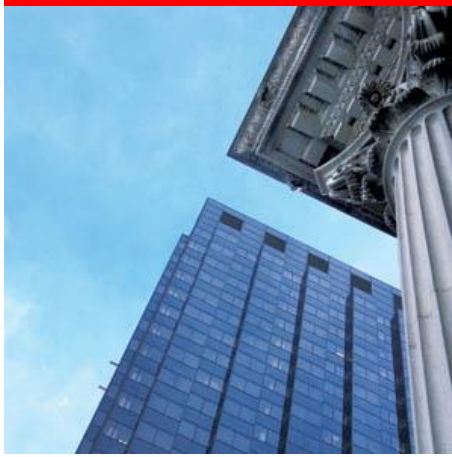
What are the some of the functionalities provided by Query Options?

Answer: Query Options provides range, sort, and propertyname pre caching functions.

Summary

- Repository API is the foundation of persistant object storage, user profiling, and content targeting in ATG.
- The core API elements of the Repository API are the Repository, RepositoryView, RepositoryItem, MutableRepository, and PropertiesChangedEvent.
- Base interfaces of a repository are immutable data store. Mutable Repository interface defines methods for CRUD operations.
- Repositories queries can be parameterized.
- Query Builder and Query Options allow for programmatic object based construction of queries.
- With named queries, developers can specify and name the query in the repository definition file and use all over the application.

Q&A



ORACLE IS THE INFORMATION COMPANY


ORACLE®