



**ORACLE®**

## **Introduction to Forms and Form Handlers**

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.

# Learning Objectives

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

- Learn about ATG's basic form handlers and how to use them.
- Understand and use DSP form tags to embed forms in JSP Pages
- Set form handler property values on the page
- Submit a form and understand the order of form processing on submission
- Use OOTB ATG form handlers to achieve common business objectives
- Extend OOTB ATG form handlers and create custom form handlers for custom requirements
- Deal with and display form errors

# Agenda

- DSP Forms
- Form Handler
- Custom Form Handlers

# Section 1: DSP Forms



# Introduction to ATG Form Processing

- Many web applications obtain information from users by having them fill out forms.
- DSP Form features:
  - Page can display a component property's value as a form element's value.
  - Input values can be written to component properties when a form is submitted.
  - Can interact directly with a SQL database.
  - Input elements can trigger actions when a form is submitted.

# DSP Form Tags

- DSP form tags:
  - Similar to their HTML equivalents.
  - Support standard HTML attributes.
  - Link form fields to component properties.
- Commonly used Form tags:
  - dsp:form
  - dsp:input
  - dsp:select



# DSP Form Basics

- An ATG form is defined by the <dsp:form> tag.
- Form elements are by dsp:input and others.
- Dsp:form tag requires you to specify an action attribute.

```
<dsp:form action="/testPages/showStudentProperties.jsp"
          method="POST">
  <p>Name: <dsp:input bean="/samples/Student_01.name"
                    type="text"/>
  <p>Age: <dsp:input bean="/samples/Student_01.age"
                    type="text" value="30"/>
  <p><dsp:input bean="/samples/Student_01.submit">
        type="submit" value="Click to submit"/>
</dsp:form>
```

# DSP Tags and HTML Conventions

- DSP Library provides tags that are similar to their HTML equivalents.
- In general they support HTML attributes. For e.g. size="25" on dsp:input is used to specify the length on the tag.
- The name attribute can be omitted. If so, ATG assigns one.
- Unlike HTML, ATG forms require explicit values for all attributes as shown below for checked attribute.

```
<input type="checkbox" name="re"
      value="primary" checked /><br />
<dsp:input type="checkbox" name="re"
          value="primary" checked="true" />
```

# Setting Property Values in Forms

- HTML forms can set the values of Nucleus component properties by using the bean attribute.
- A single form can set the properties of various nucleus components.
- In general, only one input tag in the form can be associated with a given property.
- All form tags must be inside the dsp:form tag.
- When the form submits, the values entered in the form replace the nucleus component properties.
- The following properties can be set:
  - Scalar (non-array) properties
  - Array properties
  - Map properties

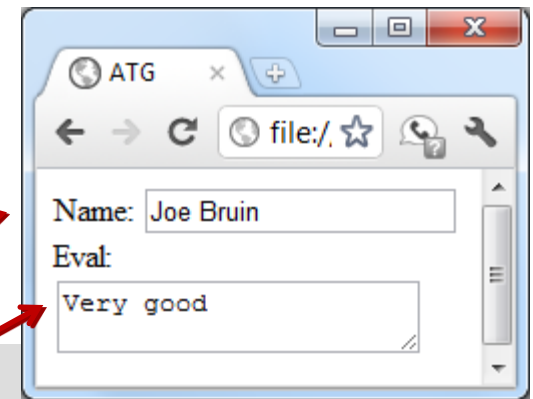
# Setting Scalar (Non-Array) Properties

- Several different form controls can set the values of scalar properties, depending on data type of the property.
- The different types are:
  - Checkboxes
  - Radio Buttons
  - Dropdown lists
  - Text Entry Fields
    - Single line
    - Multiple line areas

# Setting Scalar Properties using Text Entry Fields

- Single line text entry is achieved with **dsp:input** with type set to text.
- **dsp:textarea** creates a multi-line area for longer text entries.

```
<dsp:form action="test.jsp" method="POST">  
  Name: <dsp:input bean="/samples/Student_01.name"  
    type="text"/>  
  Eval: <dsp:textarea bean="/s/Student.eval">  
    Very good</dsp:textarea>  
</dsp:form>
```



# Setting Scalar Values using dsp:input

- When used with a bean attribute, the field is filled with its current value from the Nucleus component properties.
- When used with the value property, the text specified in value is displayed instead.
- dsp:input can also be associated with other data types such as Integer. ATG will convert it on form submission.

```
<dsp:input bean="/samples/Student_01.name" type="text"/>  
<dsp:input bean="/samples/Student_01.age"  
           type="text" value="30"/>
```

- The name text box displays the name from nucleus component and the age displays 30 (over ridden).
- When form submits, the values are written to specified properties.

# Setting Values using textarea Input Tag

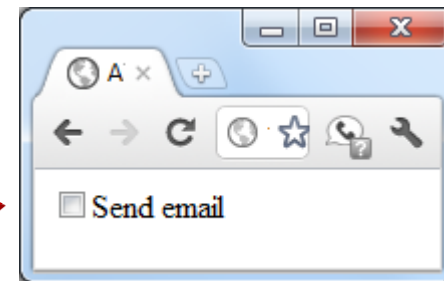
- Creates a multi-line area for longer text entries.
- Accepts standard HTML attributes such as rows and cols.
- Default values can be set with either the default attribute or between open and close tag.

```
<dsp:textarea bean="/s/Student_01.eval"></dsp:textarea>  
<dsp:textarea bean="/s/Student_01.eval"  
  default="-Enter eval here"></textarea>  
<dsp:textarea bean="/s/Student_01.eval">  
  -Enter eval here-</textarea>
```

- In the statements above, second and third are equivalent.
- If both default and value are specified, default specified value takes precedence.

# Using Checkboxes

- Can set the value of boolean property.
- The current value of the property sets the checkbox's initial display.
- It can be overridden with checked="true" attribute.
- A default value can be specified with a default tag.



```
<dsp:input bean="/s/Student.emailOK" type="checkbox" checked="true" /> Send Email
```



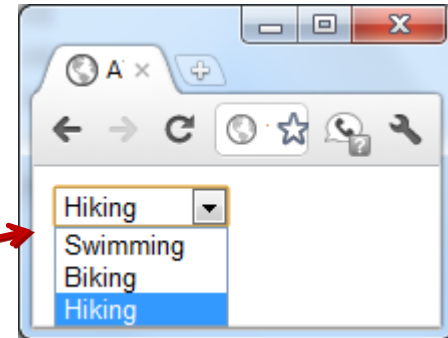
# Using Radio Buttons

- Use `dsp:input` with type `radio` to use Radio buttons.
- Grouping is based on sharing the same bean attribute.
- Within a selection group, only one input tag can set its `checked` attribute to `true`.

```
<p>Choose your favorite outdoor sport:  
<dsp:input bean="/s/Student_01.fav" type="radio"  
  value="swimming"checked="false"/>Swimming  
<dsp:input bean="/s/Student_01.fav" type="radio"  
  value="biking"checked="false"/>Biking  
<dsp:input bean="/s/Student_01.fav" type="radio"  
  value="hiking"checked="true"/>Hiking
```

# Using Dropdown Lists

- dsp:select tag is used to create a set of choices in a drop down list.
- Dsp:option tag is used to specify the options.



```
<dsp:select bean="/s/Student_01.fav">
  <dsp:option selected="false" value="swimming">
    Swimming</dsp:option>
  <dsp:option selected="false" value="biking">
    Biking</dsp:option>
  <dsp:option selected="true" value="hiking">
    Hiking</dsp:option>
</dsp:select>
```

# Setting Array Properties

- The array property can be set with either of the two:
  - Grouped set of check boxes.
  - Multiple selection list box.
- Both of the above are functionally equivalent ,but provide different layout choices.
- The Group check box is similar to check box but specify the same bean value.

```
<dsp:input bean="/s/Student.hobbies" type="checkbox"
           value="Swimming" /> Swimming
<dsp:input bean="/s/Student.hobbies" type="checkbox"
           value="Biking" /> Biking
<dsp:input bean="/s/Student.hobbies" type="checkbox"
           value="Climbing" /> Climbing
```

# Setting Array Property with Select Tag

- The dsp:select creates a multiple selection list box if its multiple attribute is set to true.
- By default ,the current values in the array are selected.
- They can be overridden with selected="true" attribute.

```
<dsp:select multiple="true" bean="/s/Student_01.hobbies">
  <dsp:option value="swimming">Swimming</dsp:option>
  <dsp:option value="biking">Biking</dsp:option>
  <dsp:option value="climbing">Climbing</dsp:option>
  <dsp:option value="photo">Photography</dsp:option>
  <dsp:option value="fencing">Fencing</dsp:option>
</dsp:select>
```

# Setting Map Properties

- Map properties can be set using the ATG's built in map converter facility on the dsp:input tag.
- The converter attribute is set to the map tag converter, which ensures that on form submission, the map settings are applied correctly to the target Map property.

```
<dsp:input bean="FormHandler.address"
  converter="map" value="street=" />
<dsp:input bean="FormHandler.address"
  converter="map" value="city=" />
<dsp:input bean="FormHandler.address"
  converter="map" value="state=" />
<dsp:input bean="FormHandler.address"
  converter="map" value="zip=" />
```

- You can also use the hidden input control to set map property.

# Set Property Values via Hidden Inputs

- A component might contain one or more properties whose values are not visible on the form.
- These are programmatically set, often computed from other use supplied values.

```
<dsp:input type="hidden"  
  bean="/samples/Student_01.lastSubmit"  
  beanvalue="/atg/dynamo/service/CurrentDate.secondAsDate"/>
```

- The statement above saves the current time to lastSubmit property on form submit.

# Submitting Forms

- The input tag for a form submit control is as follows:

```
<dsp:input type="submit" bean="/samples/Student_01.age"
  submitvalue="34" value="Click Here"/>
```

- When the form submits, the optional bean property is set by submitvalue if specified. If not, value is used.
- You can also use an image to submit the form.

```
<dsp:input bean="/samples/Student_01.emailOK"
  type="image" src="clickpic.gif" submitvalue="true"/>
```

- If submitvalue is omitted, the pixel coordinates are submitted as value.

# Order of Tag Processing

- ATG Platform assigns a default priority to all form controls.
  - Input elements of type submit and image are assigned priority of -10.
  - All other form elements are assigned priority 0.
- On form submission, elements are processed in descending order of priority.
- A low value for submit ensures that the set and handle methods execute after all other inputs are processed.
- Priority can be manually set using the priority="value" tag on any form tag.



# Section 1



## Check Your Understanding

What are some of the dsp form tags to help you set nucleus properties?

**Answer: dsp:form, dsp:input, dsp:select, etc.**

# Section 1



## Check Your Understanding

What attribute allows for the nucleus component to be specified on the dsp form tags?

**Answer: bean property**

# Section 1



## Check Your Understanding

Which types of tags can you use to set scalar (non-array) properties?

**Answer: We can use text boxes, text area boxes, check boxes, radio buttons, and drop down lists to set non array properties.**

# Section 1

## Check Your Understanding

Which types of tags can you use to set array properties?

**Answer: You can use either dsp:input type="checkbox" or dsp:select/dsp:option tags to set array properties.**

# Section 1



## Check Your Understanding

How are map properties set?

**Answer: Map properties are set using the dsp:input tag with a map tag converter.**

# Summary

- **DSP Form tags provide tags that are similar to their HTML counterparts.**
- **DSP Form tags set the component properties when a form is submitted.**
- **Form tags can set scalar values, array values and even map properties in nucleus components.**
- **The nucleus component set methods are called in descending order of priority.**
- **By default the submit method has low priority, which makes it handle the method called after all the other set methods are called.**



# Section 2:

## Form Handler



# Form Handlers Concept

- ATG platform provides form handlers that can perform the following:
  - Validate data before it is submitted.
  - Detect missing information and display appropriate messages to the user.
  - Direct users to different pages depending on form submission results.
  - Read and write database or repository data.



# OOTB Form Handler Classes

- All provided form handler classes are subclasses of `atg.repository.servlet.GenericFormHandler`.
- Handler classes:
  - `SimpleSQLFormHandler`: Works with form data that is stored in a SQL Database.
  - `RepositoryFormHandler`: Saves repository data to a database.
  - `ProfileFormHandler`: Connects forms with user profiles.
  - `SearchFormHandler`: Specifies properties available to a search engine.

# Resetting a Form

- Form handler components all support a cancel operation.
- This invokes the form's cancel button.
- The input tag is:

```
<dsp:input type="submit" value="button-label"  
          bean="handler-component.cancel"/>
```

- The cancel operation redirects the page specified in the form handler's cancelURL.
- A form handler typically provides its own implementation in the handleCancel() method.

# Form Error Handling

- A Web application must identify and gracefully handle errors that might occur on form submission.
- If the form uses a form handler class `GenericFormHandler` or one of its subclasses, exceptions are saved to one of these form handler component properties:
  - `formError`: Boolean that is set to true if any error occurs.
  - `formExceptions`: A vector of the exceptions.
  - `propertyExceptions`: A read only property that returns a dictionary of sub properties, one for each property set by the form and exceptions, if present.

# Redirecting on Form Submission

- ATG form handlers provide a number of submission operations.
- Each submission operation typically supports a pair of navigation properties like:
  - operationSuccessURL
  - operationErrorURL
- Navigation properties are set through hidden input tags.

```
<dsp:input type="hidden" bean =  
    "/atg/dynamo/droplet/MyRepositoryFormHandler.updateErrorURL"  
    value="updateStockFailure.jsp"/>
```

# Search Forms

- Search forms can help find products that satisfy a set of criteria.
- A search form handler is of class `atg.repository.servlet.SearchFormHandler`.
- The following search types are supported:
  - Keyword Search: searches string properties based on keyword.
  - Text Search: performs text pattern matching on one or more text properties.
  - Hierarchical Search: returns all parents and descendant items of a specified item.
  - Advanced Search: provides more search options.

# Search Forms Creation

- To create a search form handler:
  - Create a component in ACC with session scope based on search form handler class.
  - Configure the basic and specific properties for the type of search form.
- You must create a unique search form component for each type of search you want to use.

# Example: Keyword Search

- All search forms require you to create the following properties:
  - repositories: A comma separated list of repositories to include.
  - itemTypes: An array of item types to include in the search.
  - successURL: URL that opens when the search operation succeeds.
  - errorURL: URL that opens when a search fails.
- The Keyword Search form handler expects:
  - doKeywordSearch: enable the component for keyword search.
- The following is typically supplied by the form user:
  - keywordInput
- In addition, several other properties are available to control the search.

# Search Form Submit Operations

- The search form supports two submit operations:
  - search: launches the search.
  - clearQuery: removes all previous query data.
- Example

```
<dsp:input type="submit" bean="MySearchFormHandler.search"/>  
  
<dsp:input type="submit" bean="MySearchFormHandler.clearQuery"/>
```



# Controlling and Displaying Results

- The SearchFormHandler lets you control the display of search results and help users access them.
  - Set maximum number of items returned.
  - Set maximum number of items displayed.
  - Get number of items returned and number of result pages.
  - Set the current results page and the total number of result pages.
- After searching the form handler returns two properties:
  - searchResults: all items undifferentiated by item.
  - searchResultsByItemType: a HashMap with key value pair containing the itemType and a collection of items.
- The above can be used to either display all items or organize them by item types.

# Repository Form Handler

- Repository form handler provides methods and properties for working with repository data.
- The Repository form handler is provided by the class `atg.repository.servlet.RepositoryFormHandler`.
- This class can be used to add, update, and delete repository items.
- The benefits are:
  - Requires only repository item type for updates.
  - Supports all repository types incl. HTML, XML, LDAP, and SQL.
  - Caches repository data.
  - Optimizes data access.

# Repository Form Handler Properties

- The following are required properties:
  - repository: specifies the repository where items are to be acted on.
  - itemDescriptorName: A string that specifies the item descriptor.
- Several other properties are provided.
- More properties such as updateSuccessURL and updateErrorURL are used to control navigation.

# Repository Form Handler Submit Operations

- The following operations are supported:
  - create: creates a repository item.
  - delete: deletes a repository item.
  - update: updates a repository item.
- Example:

```
<dsp:input type="submit"
    bean="atg/dynamo/droplet/MyRepositoryFormHandler.create"
    value="Add CD"/>
```

# User Profile Forms

- User Profile Forms are used to create and modify user profiles.
- The User Profile Form Handler can be accessed thru the Nucleus component `/atg/userprofiling/ProfileFormHandler`.
- The User Profile Form Handler affects:
  - Profile creation and updates.
  - User login and logout.
  - Assignment of existing roles and orgs to individuals and groups.

# Profile Form Handler Submit Operations

- The following submit operations are supported:
  - login: validates login and password and associates the users.
  - logout: resets the profile and logs the user out.
  - changePassword: changes the user's password.
  - create: creates a permanent profile.
  - delete: deletes the current profile.
  - update: modifies the properties of the current profile.
  - cancel: cancels any changes the user made.
  - clear: clears the value dictionary.

# Profile Form Handler Example

```
First Name: <dsp:input bean="ProfileFormHandler.value.firstname"
               maxlength="30" size="25" type="text"/>
```

```
Last Name: <dsp:input bean="ProfileFormHandler.value.lastname"
               maxlength="30" size="25" type="text"/>
```

```
Email Address: <dsp:input bean="ProfileFormHandler.value.email"
                    maxlength="30" size="25" type="text"/>
```

```
<dsp:input bean="ProfileFormHandler.value.gender" type="radio"
           value="female"/>Female
```

```
<dsp:input bean="ProfileFormHandler.value.gender" type="radio"
           value="male"/> Male
```

```
<dsp:input bean="ProfileFormHandler.create"
           type="submit" value=" Save "/>
```

# Profile Form Navigation Properties

- Each submit operation has a success and error URL of the type OPERATIONSuccessURL and OPERATIONErrorURL.
- For example the create operation navigation URLs are createSuccessURL and createErrorURL.
- After the form is submitted, depending on the result, one of the two pages are displayed to the user.
- These URLs can be provided as hidden form fields in the profile form as follows :

```
<dsp:input bean="ProfileFormHandler.loginErrorURL" type="hidden"
           value="login_failed.jsp"/>
```



# Multi Profile Form Handlers

- ATG ships with two form handlers available to administrators to create and update multiple user profiles at the same time.
  - MultiProfileAddFormHandler: add multiple users at the same time.
  - MultiProfileUpdateFormHandler: updates multiple users.
- You can use these form handlers to:
  - Assign roles to a group of user profiles.
  - Assign a group of user profiles to the same organization.
  - Perform the same changes to a group of user profiles.

## Section 2



# Check Your Understanding

What form handler does ATG provide to handle user profiles?

**Answer: Profile Form Handler.**

## Section 2

# Check Your Understanding

How can the user be informed of form errors?

**Answer: ATG Generic Form Handler provides properties like `formError`, `formExceptions`, and `propertyExceptions` to inform the user of form errors.**

## Section 2

# Check Your Understanding

What are form navigation properties?

**Answer: Form navigation properties are properties used to navigate a user to another jsp after submission.**

## Section 2

# Check Your Understanding

What search types does the search form support?

**Answer: Keyword, text, hierarchical, and advanced.**

## Section 2

# Check Your Understanding

What are the key properties needed for the repository form handler?

**Answer: Repository and item descriptor name are the key fields required by the repository form handler.**

## Section 2

# Check Your Understanding

What types of operations does the profile form handler support?

**Answer: Profile Form Handler supports login/logout, profile creation, deletion, and modification.**

## Section 2

# Check Your Understanding

An Administrator would like a page to update addresses for all employees when the company is relocated. What form handler would you use?

**Answer: Multi profile form handlers allow you to make the same change across multiple users.**



# Summary

- **Form Handlers provide validation, missing information detection, processing of input data, and redirection to appropriate URL after submission.**
- **ATG ships with form handlers that can perform various operations such as repository operations, profile operations and searching.**
- **Search forms can be used for keyword search, text search, hierarchical search, and advanced search.**
- **Repository form handlers can add, update, and delete items from various repositories.**
- **User profile forms can manage login/logout, profile creation, and editing.**



# Section 3:

## Custom Form Handler



# Custom Form Handlers

- ATG ships with the following form handlers:
  - SimpleSQLFormHandler: Works with form data that is stored in a SQL Database.
  - RepositoryFormHandler: Saves repository data to a database.
  - ProfileFormHandler: Connects forms with user profiles.
  - SearchFormHandler: Specifies properties available to a search engine.
- You can extend and modify these to suit your specific requirements.
- For more complex requirements, you can also create Custom Form Handlers.

# Extending OOTB Form Handlers

- ATG OOTB Form handlers can be extended to meet specific requirements.
- As an example, let's examine the following requirement:
  - ATG OOTB Profile Form Handler uses case sensitive user name for login.
  - The client requires you to use a case insensitive login name.
- We will extend the profile form handler to achieve this result.

# Extending the Profile Form Handler

- Profile Form's handleLogin method uses the IdentityManager interface to handle user login verification.
- The following methods are called:
  - ProfileForm.findUser()
  - ProfileFormHandler.preCreateUser()
- We will intercept the preCreateUser call and convert the login name to lowercase before creation.
- We will intercept the findUser call and convert the login name to lowercase before searching.
- The API reference provides all available methods and properties.

# Extend the Class (1)

- Extend ProfileFormHandler and override the method preCreate User. Call it MyProfileFormHandler.java.

```
protected void preCreateUser(DynamoHttpServletRequest pRequest,
                             DynamoHttpServletResponse pResponse)
    throws ServletException, IOException {
    super.preCreateUser(pRequest, pResponse);
    String memberName = getValue().get("memberName"); // get name
    if (memberName != null) {
        String login = memberName.toLowerCase(); // convert to lower
        getValue().put("login", login); // put it back
    }
    else {
        // If the member name is not available, then make
        // sure we clear out any old login values
        getValue().remove("login");
    }
}
```

# Extend the Class (2)

- Extend the class ProfileForm and override the method findUser. Call it MyProfileForm.java.

```
protected RepositoryItem findUser(String pLogin,String pPassword,
    Repository pProfileRepository, DynamoHttpServletRequest
    pRequest, DynamoHttpServletRequestResponse pResponse)
    throws RepositoryException, ServletException, IOException {
    if (pLogin != null) {
        return super.findUser(pLogin.toLowerCase(), pPassword,
            pProfileRepository, pRequest, pResponse);
    }
    else {
        return super.findUser(pLogin, pPassword,
            pProfileRepository, pRequest, pResponse);
    }
}
```

# Override the Class in Nucleus

- ProfileFormHandler is already registered with the Nucleus in an OOTB installation.
- Your property file will layer the nucleus component and override the class definition.
- Ensure that your module appears after the OOTB modules in CONFIG PATH.
- ProfileFormHandler.properties should be placed at /atg/userprofiling/

```
#/atg/userprofiling/ProfileFormHandler  
$class=MyProfileFormHandler
```

- ProfileForm.properties should be placed at /atg/userprofiling

```
#/atg/userprofiling/ProfileForm  
$class=MyProfileForm
```



# Custom Form Handlers

- ATG form handler classes all implement the interface `atg.droplet.DropletFormHandler`.
- Three form handler bases classes are provided that implement this interface.
  - `atg.droplet.EmptyFormHandler`: provides empty implementation of methods.
  - `atg.droplet.GenericFormHandler`: extends `EmptyFormHandler`. Provides simple implementation and adds error handling.
  - `atg.droplet.TransactionFormHandler`: extends `GenericFormHandler`. Treats the form handling process as a transaction.

# Creating a Custom Form Handler

- The following are the steps for creating a custom form handler:
  - Extend GenericFormHandler.
  - Implement get and set methods for input and output properties.
  - Implement handle methods for form actions.
  - Implement standard form error handling.
- Register the form handler with Nucleus:
  - Create a property file for the Form Handler.
  - Specify any configuration parameters.
- Embed the form handler on the page:
  - Embed the form handler on the page.
  - Show errors if any.
  - Show response if no errors.

# Creating the Form Handler

- We will create a form handler to check the balance of a loyalty point card.
- Extend Generic Form Handler and create MyLoyaltyFormHandler.
- Skeleton is shown here:

```
public class MyLoyaltyFormHandler
    extends GenericFormHandler {
    // Define private properties
    // Define get and set methods
    // Define handle methods
}
```

# Define Get and Set Methods

- Define any private variables and get/set methods.

```
public class MyLoyaltyFormHandler
    extends GenericFormHandler {
    private String  cardNumber;
    private String  pointsResponse;

    public void setCardNumber(String cardNumber) {
        this.cardNumber = cardNumber;
    }
    public String getCardNumber() {
        return this.cardNumber;
    }
    public void setPointsResponse(String pointsResponse) {
        this.pointsResponse = pointsResponse;
    }
    public String getPointsResponse() {
        return this.pointsResponse;
    }
    // Additional get set for WS Client
    // Define handle methods
}
```

# Handler Method (1)

- Form Handlers use HandleX methods to link form elements with Nucleus components.

```
public boolean handleX (  
    javax.servlet.http.HttpServletRequest request,  
    javax.servlet.http.HttpServletResponse response )
```

- Handle method can also throw exception `java.io.IOException` and `javax.servlet.ServletException`.
- Handle method can also use ATG extensions for request and response.

```
public boolean handleX (  
    atg.servlet.DynamoHttpServletRequest request,  
    atg.servlet.DynamoHttpServletResponse response )
```

# Handler Method (2)

- Handler Methods return:
  - True, normal processing of the remaining values continues.
  - False, no further values are processed after the handler is called.
- Handle methods are called on form submission.
- Handle methods can also be called after setX method.
- Submit Handler Methods are powerful as:
  - They are associated with a form's submit button.
  - Processed after tags that use other input types.
- A form handler can have multiple submit handler methods.

# Define the Handle Method

- The Handle Method accepts requests and responses, and throws the relevant exceptions.
- Note that the errors are added to Form exceptions.
- Check form redirect is provided by Generic Form Handler.

```
public boolean handleCheckPoints(  
    DynamoHttpServletRequest request,  
    DynamoHttpServletResponse response)  
    throws ServletException, IOException {  
    try {  
        pointsResponse = WSClient.wsCheckPoints(cardNumber);  
    } catch (Exception e) {  
        addFormException(new DropletException(  
                                e.getMessage(), e));  
    }  
    return checkFormRedirect(  
        request.getRequestURL().toString(),  
        request.getRequestURL().toString(),  
        request, response);  
}
```

# Register the Form Handler

- Form handler should be registered as a Nucleus component.
- The property file can be defined as:

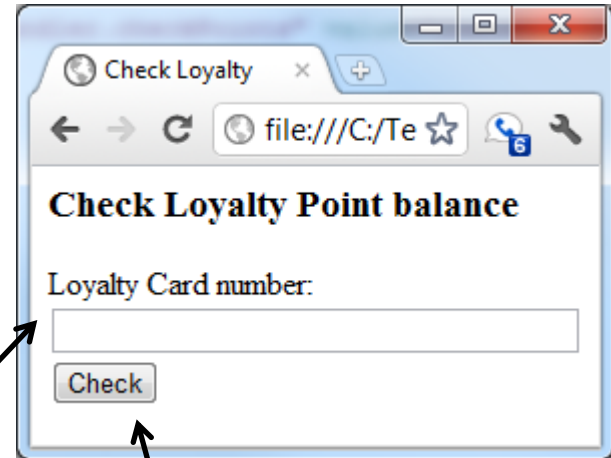
```
#path /myProject/form/MyLoyaltyFormHandler  
$class=com.myproject.forms.MyLoyaltyFormHandler  
$scope=request  
WSCliet=/gateway/loyalty/WSCliet
```

- Form Handlers can be request or session scoped.
- Use request scoped for most form handlers.
- Session scoped form handlers can be used for multi page forms.
- Special care must be used with session scoped form handlers to avoid data corruption.



# Embed the Form Handler on the Page

- Use `dsp:form` and `dsp:input` tags to embed form on page.
- Link to component properties with bean attribute.



```
<dsp:form id="loyaltyForm" formid="loyaltyForm"
          method="post" action="myCheckPoints.jsp">
  Loyalty Card number:
  <dsp:input type="text" size="38"
            bean="MyLoyaltyFormHandler.cardNumber"
            required="true" value=""/>
  <br />
  <dsp:input type="submit"
            bean="MyLoyaltyFormHandler.checkPoints"/>
  <br />
</dsp:form>
```

# Form Exceptions

- The handle method should catch exceptions and add them to form exceptions in a user readable format.

```
addFormException(new DropletException("Card is not valid", e));
```

- FormError is set to true or false based on whether exceptions occurred during submission.
- Property exceptions return a map of exceptions keyed off of the property that generated the exception.
- On the JSP Page use the droplet ErrorMessageForEach to iterate and display the exceptions.

# Displaying Form Exceptions

- Use Switch droplet to check formError.
- If true use ErrorMessageForEach droplet.

```
<dsp:droplet name="Switch">
  <dsp:param bean="MyLoyaltyFormHandler.formError"
              name="value"/>
  <dsp:oparam name="true">
    <dsp:droplet name="ErrorMessageForEach">
      <dsp:param name="exceptions"
                  bean="MyLoyaltyFormHandler.formExceptions" />
      <dsp:oparam name="output">
        <LI><dsp:valueof param="message"/>
      </dsp:oparam>
    </dsp:droplet>
  </dsp:oparam>
</dsp:droplet>
```

# Meeting Business Requirements with Form Handlers

- Most dynamic web sites require the application to collect user input and process it to satisfy business requirements.
- ATG Developers should use form handlers in this order:
  - OOTB ATG Form Handlers
  - Extensions to OOTB ATG Form Handlers
  - Custom Form Handlers
- Note that while extending OOTB ATG Form Handlers, you only need to override the specific functionality that is different.
- The API documentation provides details on each of the form handlers and their properties and methods.

## Section 3

# Check Your Understanding

What form handlers does ATG ship with?

**Answer: SimpleSQLFormHandler,  
RepositoryFormHandler,  
ProfileFormHandler, and  
SearchFormHandler.**

## Section 3



# Check Your Understanding

How do you extend an OOTB ATG Form handler?

**Answer: Extend the class, override the method, and register the class with nucleus.**

# Section 3

## Check Your Understanding

How do you create a custom form handler?

**Answer: Create the class by extending GenericFormHandler, implement get/set and handle methods, and register the class with nucleus as a component.**

## Section 3



# Check Your Understanding

How do you satisfy a complex form handler business requirement?

**Answer: First see if an OOTB form handler will do the job. If not, extend an OOTB form handler. If that is not possible, create a custom form handler.**



## Section 3

# Check Your Understanding

What facilities does ATG provide to display form errors to the user?

**Answer: ATG provides `formError`, `formExceptions`, and `propertyExceptions` properties on Generic Form Handler. These can be used to display form errors to the user.**

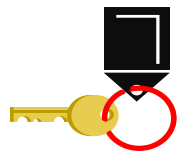
# Summary

- **Form Handlers accept user input, process them, and provide meaningful responses to the users.**
- **ATG Ships with OOTB form handlers such as repository form handler, profile form handler, and search form handler.**
- **These can be extended to satisfy specific requirements.**
- **More complex requirements can be met with custom form handlers.**
- **To extend a OOTB form handler, extend the class, override the method, and register the class with nucleus.**
- **To create a custom form handler, create the class, implement get/set and handle methods, and register the class with nucleus.**
- **Form handlers provide mechanisms to inform the user of form exceptions.**



# Key Points

- Nucleus is a key ATG feature for building component based web applications.
- Simple JavaBeans can be combined with configuration property files to make components.
- Components can have global, session, request, or window scope.
- Configuration property files have to be specified in the CONFIGPATH.
- ATG will smartly layers and merges property files for components.
- GenericService implements useful Nucleus interfaces to add additional functionality.





ORACLE IS THE **INFORMATION** COMPANY

ORACLE®