Web Services Customization Guide

Web Services Customization

This guide will describe the steps for creating custom modifications for WM web services, along with the proper steps for each type of customization. For more in-depth implementation details of WM Web Services, please refer to the how-to guide.

Initial Setup

This section outlines the steps required before customizations will be properly deployed.

%LESDIR%/ws/webapp/web.xml

In order to provide web service customizations, one must create the web.xml file in the path specified above. This file will automatically be placed into all WAR files with LES customizations.

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>SpringDispatchServlet</servlet-name>
    <display-name>Spring Dispatch Servlet</display-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
      <param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/les-webservices.xml,/WEB-INF/spring/webservic
es-config.xml</param-value>
    </init-param>
  </servlet>

  <filter>
    <filter-name>Authentication</filter-name>

<filter-class>com.redprairie.moca.servlet.AuthenticationFilter</filter-cla
ss>
    <description>
       This filter does standard authentication
    </description>
  </filter>

  <filter-mapping>
    <filter-name>Authentication</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <servlet-mapping>
    <servlet-name>SpringDispatchServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

%LESDIR%/ws/wm/spring/les-webservices.xml

Any webservice customizations will require modifications to spring. An import of webservices.xml is required so that existing WM Spring definitions may be modified.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- Import WM standard mappings -->
    <import resource="webservices.xml" />

</beans>
```

Custom Code

Web Resources

WebResources are classes representing data objects exchanged with web services. There are two generic types of WebResources that provide tools for common functionality:

- **AbstractWebResource**: Used for a resource that doesn't need to be modified or deleted.
- **AbstractIdentifiableWebResource**: Used for resources identifiable by a WebResourcePK, typically supports all CRUD operations.

The java customizations for resources should be deployed in **%LESDIR%/src/java/com/redprairie/les**.

A potential problem that one may encounter is adding custom columns for a table that already has a web resource associated with it. The easiest solution is to create a custom resource class that extends the old resource's class. Assume this extension was named CustomFoo, the spring mapping would be the following:

```
<bean id="aVerySpecialFoo"
class="com.redprairie.les.web.services.examplepackage.resources.CustomFoo"
scope="prototype"/>
```

Managers

The manager's responsibilities consist of performing operations for controllers.

- **AbstractWebResourceManager**: This generic class outlines the methods needed for managers belonging to AbstractWebResources. It expects that the user, at the very least, would like to list a collection of specific resources.
- **AbstractIdentifiableWebResourceManager**: This generic class outlines the methods needed for managers belonging to AbstractIdentifiableWebResources. It expects that the user would like to view, modify, create, or delete individual resources (as uniquely identified by their primary key).

The java customizations for managers should be deployed in **%LESDIR%/src/java/com/redprairie/les**.

Creating/Modifying Manager commands

If FooManagerImpl inherits from AbstractIdentifiableWebResourceManager, the commands it executes can be rewired without needing to modify the associated Java class.

```
<bean id="fooManager"
class="com.redprairie.les.web.services.examplepackage.managers.FooManagerI
mpl">
    <property name="listCommand" value="list foos where @+bar" />
    <property name="deleteCommand" value="remove foo where @+bar" />
</bean>
```

The command definitions supplied in this bean definition would take precedence over any previously assigned commands for FooManagerImpl.

Modifying Column Configuration

It may be possible that one may want to override an existing column configuration for a manager that is using an LSQuery. In order to do this, there must first be .json file (such as customColumnConfiguration.json) in **%LESDIR%/data/rpuxQueryColumnMappings** (for new column mappings) or **%DCSDIR%/data/rpuxQueryColumnMappings** (for existing column mappings)

```
<bean id="fooManager"
class="com.redprairie.les.web.services.examplepackage.managers.FooManagerI
mpl">
    <property name="listCommandColumnMapping"
value="customColumnConfiguration" />
</bean>
```

Wiring Custom Managers

If one wanted to change the underlying logic for the manager from the previous section without rewriting controller logic, he or she would simply write a class implementing the FooManager interface, such as FooManagerCustomImpl

```
<bean id="fooManager"
class="com.redprairie.les.web.services.examplepackage.managers.FooManagerC
ustomImpl" />
```

Controllers

The controller's role is to take HTTP web service requests and direct them to a place responsible for handling them.

WM gives abstract controller classes for easier implementation:

- **AbstractController**: This controller provides functionality such as generating HTTP response statuses for exception handling. It also has a utility for including/excluding properties returned in requests (more information can be found at http://confluence.jda.com/display/wms/Client+driven+views+for+RESTful+Web+Resources).
- **AbstractWebResourceController**:
  This generic controller further reduces boilerplate code. If using the manager functionality provided, only 2 methods have to be implemented:
  - setBaseResourceManager(): A method that wires the manager to perform its operations.
  - getBaseResourcePath(): Returns the String for the URI extension for directing requests.

NOTE: If the WM managers aren't being used with the AbstractWebResourceController, list() and listResources() must be overridden.

- **AbstractIdentifiableController**: This is for controllers dealing with resources uniquely identifiable by their primary key classes. If using the manager functionality provided, only 2 methods have to be implemented:
  - setBaseResourceManager(): A method that wires the manager to perform its operations.

- getBaseResourcePath(): Returns the String for the URI extension for directing requests.

__NOTE__: If the WM managers aren't being used with the AbstractIdentifiableController, getResource(), postResource(), putResource(), and deleteResource() must all be overridden.

Java customizations for controllers must be deployed in **%LESDIR%/ws/wm/src**. After creating the custom controller, one must add a mapping to les-webservices.xml for Spring to detect a custom controller, as demonstrated below:

```
<bean id="fooController"
class="com.redprairie.les.web.services.examplepackage.controllers.FooContr
oller" />
```

If a bean for "fooController" already exists, it will be overwritten with the class given in the %LESDIR% Spring mapping.

Creating/Overwriting MocaComponentResource Commands

If a command was originally just listing foos without filter, and it is now decided that the webservice results should be able to filter based on a specified bar.
Also, the foos return a ship-from state and ship-to state that one may want descriptions for. The les-webservices.xml override would do something like the following:

```
<bean id="listFoos"
class="com.redprairie.les.web.services.legacy.resources.MocaComponentResou
rceImpl" >
    <property name="mocaCommand" value="list foos where @+bar" />
    <property name="descriptionIncludes" ref="fooDescriptionIncludes">
        <map>
            <entry key="ctry_name|sf_state" value="adrstc" />
            <entry key="ctry_name|st_state" value="adrstc" />
        </map>
    </property>
</bean>
```

Note: The key represents the columns from the MOCA result sets that are used to build the dscmst column value from. The value represents the dscmst column name to lookup.

Deploying Customizations

After all products have been built and %LESDIR% successfully compiles, one must run **%LESDIR%/bin/wsdeploy.exe** in order to apply customizations to WAR files.