Error rendering macro 'add-label' : Cannot invoke method replaceAll() on null object

Web Services Programmer's Guide

Version 1.4

April 27, 2018

Legal Notice

COPYRIGHT © 2015, JDA SOFTWARE GROUP, INC. ALL RIGHTS RESERVED. JDA IS A REGISTERED TRADEMARK OF JDA SOFTWARE GROUP, INC. ALL OTHER COMPANY AND PRODUCT NAMES MAY BE TRADEMARKS, REGISTERED TRADEMARKS OR SERVICE MARKS OF THE COMPANIES WITH WHICH THEY ARE ASSOCIATED. JDA RESERVES THE RIGHT AT ANY TIME AND WITHOUT NOTICE TO CHANGE THESE MATERIALS OR ANY OF THE FUNCTIONS, FEATURES OR SPECIFICATIONS OF ANY OF THE SOFTWARE DESCRIBED HEREIN. JDA SHALL HAVE NO WARRANTY OBLIGATION WITH RESPECT TO THESE MATERIALS OR THE SOFTWARE DESCRIBED HEREIN, EXCEPT AS APPROVED IN JDA'S AGREEMENT WITH AN AUTHORIZED LICENSEE.

Acronym and Keyword Directory

- **DS** - Development Services – the engineers and their leadership responsible for enhancing standard JDA solutions. The DS team delivers New Features to satisfy customer specific requirements.
- **CDP** – Customer Development Process – The DS process that is used to deliver custom development to JDA's standard software.
- **PRS** – Product Request Specification – The PRS Form provides direction and documentation of a customer's business problem that needs to be solved.
- **GDS/ GSE** - Global Delivery Services/ Global Solutions Enablement – the global solution specialists and their leadership narrowly and deeply focused on the Market Leadership and Invest products within the Solution Suites
- **CoE** - Center of Excellence – the Consulting Services associates based in India and Poland and their leadership trained, skilled, and experienced
- **JIRA** – JIRA Workflow System – is a software management tool used by JDA to manage the CDP process workflow.
- **FA** – Functional Approach – Customer facing document created by DS to summarize the functional requirements and provide a solution approach for the New Feature.
- **DD** – Detailed Design – Internal document created by DS detailing the technical solution. This document describes the software changes at a technical level.
- **PM** – Project Manager – the Regional Consulting professional accountable for the end-to-end management of schedule, scope, budget, and quality of a Customer-facing project
- **CP** – Client Partner – the Regional Consulting professional accountable for managing the Customer relationship to further exploit future opportunities for JDA.
- **RA** – Risk Assessment / Statement – the document provided by the DS team which details the risk of pursuing the modification. This document should be reviewed with the client to determine if they wish to pursue the modification. May be referred to as Risk Assessment or Risk Statement.
- **QA** – Quality Assurance – is a way of preventing mistakes, defects and to verify that features and functionality meet business objectives, and that code is relatively bug free prior to shipping or releasing new software products and versions
- **QE** – Quality Engineering – is an ongoing process focusing on quality control and quality assurance management through use of standards, statistical tools, performance metrics and various other disciplines to improve the output of our processes

Contents

About This Guide

The purpose of this guide is to describe how to integrate custom web services to PrIME. General web service development is outside the scope of this document as it assume the reader is familiar with web service concepts. See the MOCA Developer Guide further guidance on developing components in Java.

Applicability

This Guide is applicable to the following PrIME releases.

PRFC	Release
PRFC660	R3.2
PRFC707	R4.1
PRFC776	R4.1

Overview

Custom web services can be developed by JDA Development Services to implement a change request from P&G. Custom web services can also be developed by a third party, for example HP, Oracular, or P&G themselves. This architecture allows custom web services to be deployed through the standard rollout mechanism, much the same as deploying a custom MOCA command.

A predefined directory structure is used for custom web services. By placing the web service in the appropriate directory and building the system, the web service is automatically made available. In addition to the already existing web services, the following structure under $LESDIR is provided.

ws/varint_auth - Authenticated web services developed by JDA

ws/varint_nonauth - Non-authenticated web services developed by JDA

ws/usrint_auth - Authenticated web services developed by a third party

ws/usrint_nonauth - Non-authenticated web services developed by a third party

Technical Details

The build.xml file uses `ant`. All web services in the deployed directory structure are built/rebuilt and the war files placed in the appropriate directory. The `ant` command is triggered by executing `make` from the $LESDIR. `make` is called using the `REBUILD LES` directive in the rollout or by manually entering it at the command line during development.

Web services must be developed in Java.

Testing the Examples

Within each distributed directory structure, a default "Hello World" web service is distributed as a template. These simple example web services respond to both a GET and a POST.

Web Services without Authentication

The GET method can be tested using a web browser. Enter the path to the web service you wish to test. For example, to test the usrint_nonauth Hello World example, enter `http://host:port/ws/usrint_nonauth/hello_world` where `host` and `port` are the connection url to the MOCA server. For example, if the MOCA server is running on machine md1svcvlnx21 port 4600, you would enter http://md1svcvlnx21:4600/ws/usrint_nonauth/hello_world If the MOCA server uses a secure connection, use `https` instead of `http`.

The example application responds with "`Connected to the Hello World web service in usrint_nonauth`".

The POST method requires a client which can send a POST request. Many tools are available that do this, for example Postman which is available at https://www.getpostman.com/

Web Services with Authentication

Testing web services requiring authentication requires a call to first to the MOCA login webservice. This webservice is located at http://host:port/ws/auth/login and has both a POST and GET method. In both cases the usr_id and password need to be passed along to it.

For a GET request, the user ID and password in the URL. For example, http://host:port/ws/auth/login?usr_id=super&password=super

No body is required for the GET request. If submit a GET request through the browser with an invalid user name or password, the login web service responds with `HTTP ERROR 500 User login is invalid`.

For a POST request, do not pass the user ID and password in the URL. For example, http://host:port/ws/auth/login A body is required for the POST request. The body contains the user ID and password, for example:

```
{
    "usr_id" : "super",
    "password" : "super"
}
```

In both cases a MOCA-WS-SESSIONKEY cookie is returned. This cookie contains a MOCA session key, and needs to be passed to all Authenticated web service calls.

> Standard Product 2014.1 support web services authentication but not authorization. As long the client supplies a valid user name and password, the client can access any web service.

When using a browser, the browser must be enabled to store cookies. When using a tool to send POST requires, configure the tool to store and use cookies.

Developing a New Web Service

This section describes how to develop a new web service.

First, determine if the web service needs authentication or not. The advantage of authentication is that only users or applications which have valid credentials on the server can use the service. The disadvantage is that the client must handle the MOCA-WS-SESSIONKEY cookie and pass it on all authenticated web service calls. The session key contained in the cookie is valid until the MOCA server is restarted or the idle timeout period expires. The idle timeout period is defined by the `session-key-idle-timeout` value in the registry. See the MOCA Developer Guide for more information. The client must be able to request a new session key when the session times out. If the session key is missing or no longer valid, the server responds with `HTTP ERROR 401 Unauthorized`.

Windows Prerequisites

To compile on Windows, Apache Ant 1.8.1 or later is required. It is available from ant.apache.org. The following is a check list to guide the installation.

```
[ ] Extract the contents of the zip file to c:\Program Files.

[ ] Rename c:\Program Files\apache-ant-1.x.y to c:\Program Files\ant

[ ] Add the ANT_HOME environment variable to your system.

    Start->Control Panel->System

    Click on the "Advanced" tab

    Click on "Environment Variables"

    Click on "New" in the "System Variables" group box

    Variable name: ANT_HOME

    Variable value: c:\Program Files\ant

    Click "OK" all the way out.

[ ] Add the ANT bin directory to your system path.

    Start->Control Panel->System

    Click on the "Advanced" tab

    Click on "Environment Variables"

    Double-click on "Path" from the "System Variables" group box

    Add %ANT_HOME%\bin to the search path

    Click "OK" all the way out.


[ ] Add the ANT bin directory to env.bat.

    Launch a fresh cmd window.
Navigate to the LES directory.

    Execute env.bat

    Execute "servicemgr -e <environment name> dump"
```

Test the above steps by starting a fresh cmd window, execute env.bat, and then execute "ant" from the %LESDIR%. It will build the java files.

If you do not want to compile on the target Windows machine containing installed JDA products, you may build the .war files on a development machine and distirbute them to the %LESDIR%\webdeploy folder on the target environment.

A Web Service without Authentication

Create a subdirectory under the $LESDIR/ws/usrint_nonauth directory. The directory naming convention is the subdirectory is the name of the web service, with subdirectories under src for the Java package name. For example, let's say your company name is Acme and you are developing a web service called `hello acme world`, you would place it under `ws/usrint_nonauth/hello_acme_world`

> Because all third party developed web services go under the same directory, it is recommended to include the company name in the name of the web service to avoid naming conflicts between third parties. Note that the java package must be `com.redprairie`.

Create the webservices.xml file under `/ws/usrint_nonauth/hello_acme_world/spring` to map the web service to the Java class.

```
<?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">
    <bean
class="com.redprairie.usrint_nonauth.hello_acme_world.hello_acme_worldWS"/
/>
</beans>
```

Create the Java file under `ws/usrint_nonauth/hello_acme_world/src/java/com/redprairie/usrint_nonauth/hello_acme_world`. In the example below, the hello_acme_worldWS responds with a text message to GET and POST requests.

```
package com.redprairie.usrint_nonauth.hello_acme_world;
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.json.JSONException;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.redprairie.moca.MocaException;
import com.redprairie.moca.MocaResults;
import com.redprairie.moca.util.MocaUtils;
/**
 * This class is used to test web services.  It responds to GET and POST
requests with a text message.
 *
 */
@Controller
```

```
public class hello_acme_worldWS
{
    /**
     * This method is used to consume a GET HTTPRequest.
     *
     * @param request
     * @param response
     * @throws IOException
     * @throws JSONException
     */
    @RequestMapping(value="hello_acme_world", method=RequestMethod.GET)
    public void getHelloWorld(HttpServletRequest request,
HttpServletResponse response) throws IOException, JSONException {
        response.setContentType("text/plain");
        response.getWriter().write("Hello World from the Acme
Corporation!");
        response.getWriter().flush();
        response.getWriter().close();
    }
    /**
     * This method is used to consume a POST HTTPRequest.
     *
     * @param request
     * @param response
     * @param argMap
     * @throws IOException
     * @throws JSONException
     */
    @RequestMapping(value="hello_acme_world", method=RequestMethod.POST)
    public void postHelloWorld(HttpServletRequest request,
HttpServletResponse response, @RequestBody String argMap) throws
IOException, JSONException {
        getHelloWorld(request, response);
    }
}
```

In order to run a MOCA command through a non-authenticated web service, the command needs to be marked as insecure. Use the `<insecure>yes</insecure>` tag in the MOCA command file to specify this.

Rebuild the system using `make` at the $LESDIR and bounce the MOCA server. The web service is now available at http://host:port/ws/usrint_nonauth/hello_acme_world

A Web Service with Authentication

Create a subdirectory under the $LESDIR/ws/usrint_auth directory. The directory naming convention is the next directory is the name of the web service, with subdirectories under src for the Java package name. For example, let's say your company name is Acme and you are developing a web service called `list acme inventory` you would place it under `ws/usrint_auth/list_acme_inventory/src/java/com/redprairie/usrint_auth/list_acme_`inventory

Because all third party developed web services go under the same directory, it is recommended to include the company name in the name of the web service to avoid naming conflicts between third parties. Note that the java package must be `com.redprairie`.

Create the webservices.xml file under `/ws/usrint_auth/list_acme_inventory/spring` to map the web service to the Java class.

```
<?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">
    <bean
class="com.redprairie.usrint_auth.list_acme_inventory.ListAcmeInventoryWS"
/>
</beans>
```

Create the java file under `ws/usrint_auth/list_acme_inventory/src/java/com/redprairie/usrint_auth/list_acme_inventory` In this example, the ListAcmeInventoryWS class returns the load number and part number from list inventory for a user specified location and optionally qualified by the part number. Optional parameters are defined in the web service using the `required=false` directiv e.

The parameters to the web service can be specified in the GET request, for example

http://host:port/ws/usrint_auth/list_acme_inventory?stoloc=STG0016&prtnum=81254177

The parameters to the web services using the POST method are passed in the body of the message, and are parsed by the web service using the format defined for the web service (JSON, XML, etc.).

Example java code is shown below.

```
package com.redprairie.usrint_auth.list_acme_inventory;
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.json.JSONException;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import com.redprairie.moca.MocaException;
import com.redprairie.moca.MocaResults;
import com.redprairie.moca.MocaContext;
import com.redprairie.moca.util.MocaUtils;
/**
 * This class is used to test web services.  It lists inventory for a
location.
 *
 */
@Controller
public class ListAcmeInventoryWS
{
    /**
     * This method is used to consume a GET HTTPRequest.
     *
     * @param request
     * @param response
  * @param stoloc
     * @throws IOException
     * @throws JSONException
     */
    @RequestMapping(value="list_acme_inventory", method=RequestMethod.GET)
    public void getListAcmeInventory(HttpServletRequest request,
         HttpServletResponse response,
         @RequestParam(value="stoloc")String stoloc,
         @RequestParam(value="prtnum",required=false)String prtnum) throws
IOException, JSONException {
     MocaContext _moca = MocaUtils.currentContext();

  String cmd = "";
  cmd = "list inventory where stoloc = '" + stoloc + "' ";
  if (prtnum != null)
  {
   if (prtnum.isEmpty() == false)
   {
    cmd += " and prtnum = '" + prtnum + "' ";
   }
  }

  try {

   MocaResults rs = _moca.executeCommand(cmd);
            while (rs.hasNext()) {

                rs.next();
                // Write the warehouse map as part of the response message
    response.setContentType("text/plain");
    response.getWriter().write(rs.getString("lodnum"));
    response.getWriter().write(" ");
    response.getWriter().write(rs.getString("prtnum"));
    response.getWriter().write("\n");

            }
        }
        catch (MocaException e) {
   response.setContentType("text/plain");
   response.getWriter().write("Error in list acme inventory web service in
usrint_auth: ");
   response.getWriter().write(e.getMessage());
        }
        response.getWriter().flush();
        response.getWriter().close();
    }

    /**
     * This method is used to consume a POST HTTPRequest.
     *
     * @param request
     * @param response
     * @param argMap
     * @throws IOException
     * @throws JSONException
     */
    @RequestMapping(value="list_acme_inventory", method=RequestMethod.POST)
    public void postListAcmeInventory(HttpServletRequest request,
HttpServletResponse response, @RequestBody String argMap) throws
IOException, JSONException {

  response.setContentType("text/plain");
  response.getWriter().write("Left as an exercise for the reader. Body of
POST message must be parsed to get the parameters.");
```

```
    }
}
```

Rebuild the system using `make` at the $LESDIR and bounce the MOCA server. The web service is now available for use.

Role based Web Service Authorization

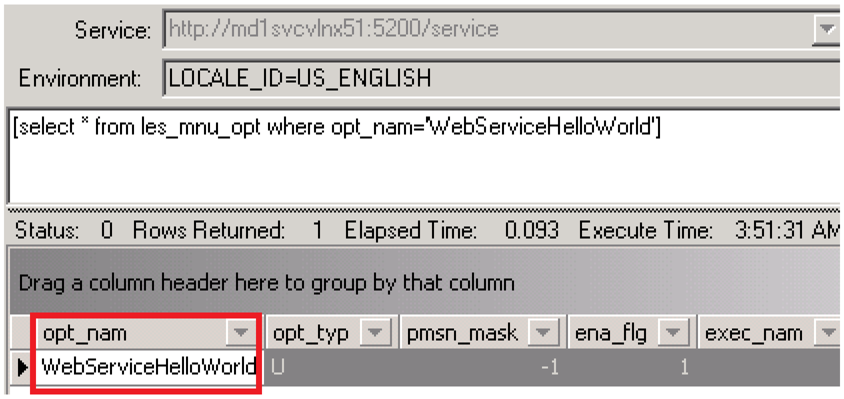
To control a user's access to a web service endpoint, MOCA provides the `@Authorization` Java annotation. It has one element, `options`,which holds role options as defined in the Role Maintenance application or a static variable.

```
package com.redprairie.moca.servlet.authorization;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(value={ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
public @interface Authorization {
    String[] options();
}
```

The `@Authorization` Java annotation is utilized in the following web service with role option `WebServiceHelloWorld.`

```
package com.redprairie.varint_auth.hello_world;

import java.io.IOException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.json.JSONException;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.redprairie.moca.servlet.authorization.Authorization;

@Controller
@Authorization(options={"WebServiceHelloWorld"})
public class HelloWorldWS
{


}
```

The role option will be made available by making following entries .



Service: http://md1svcvlnx51:5200/service

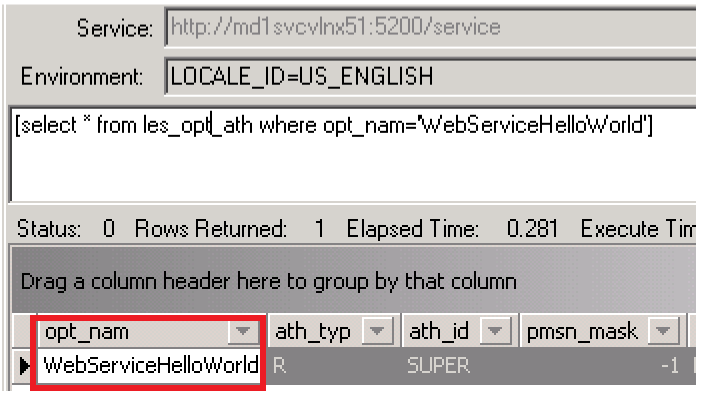
Environment: LOCALE_ID=US_ENGLISH

```
[select * from les_mnu_opt where opt_nam='WebServiceHelloWorld']
```

Status: 0 Rows Returned: 1 Elapsed Time: 0.093 Execute Time: 3:51:31 AM

Drag a column header here to group by that column

opt_nam	opt_typ	pmsn_mask	ena_flg	exec_nam
WebServiceHelloWorld	U	-1	1	



Service: http://md1svcvlnx51:5200/service

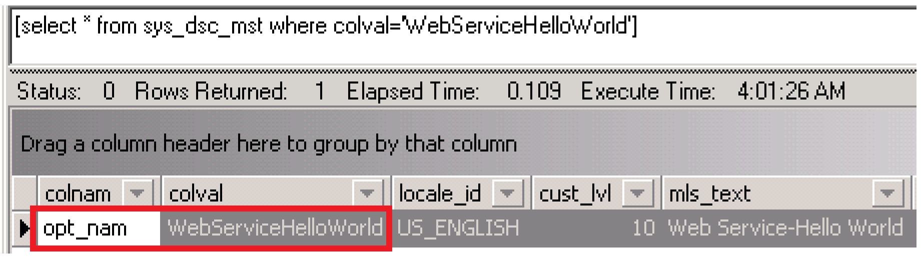
Environment: LOCALE_ID=US_ENGLISH

```
[select * from les_opt_ath where opt_nam='WebServiceHelloWorld']
```

Status: 0 Rows Returned: 1 Elapsed Time: 0.281 Execute Tim

Drag a column header here to group by that column

opt_nam	ath_typ	ath_id	pmsn_mask
WebServiceHelloWorld	R	SUPER	-1



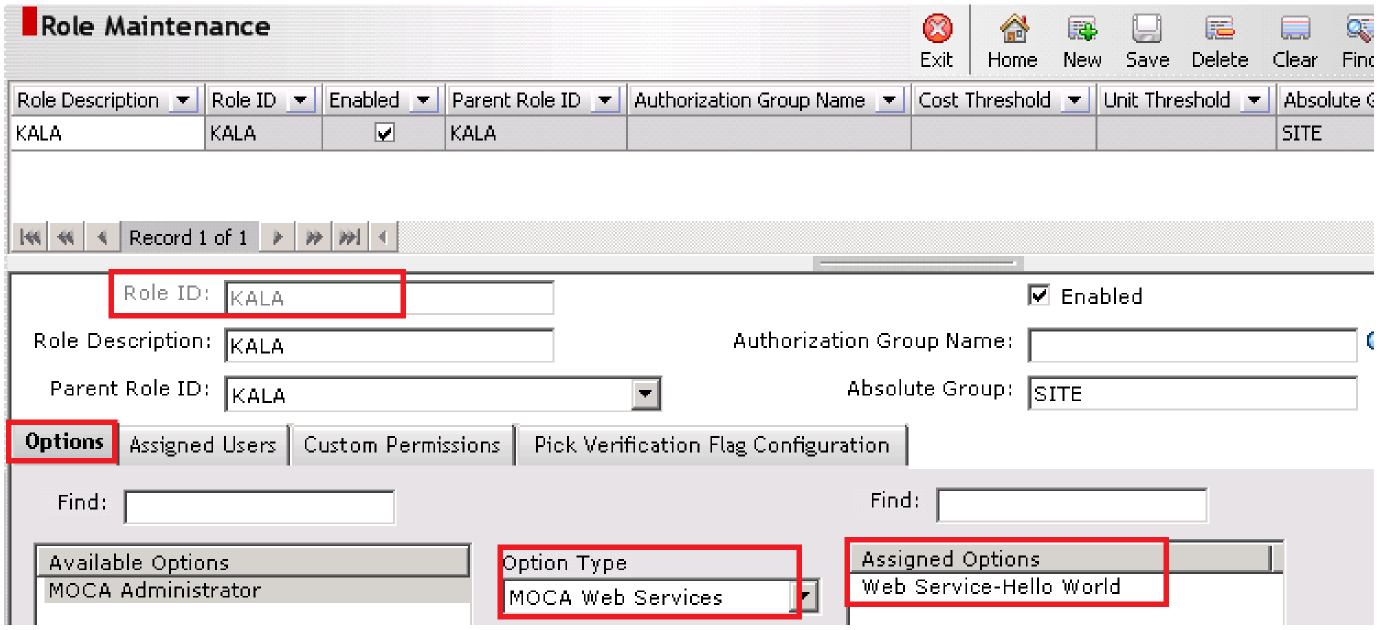
```
[select * from sys_dsc_mst where colval='WebServiceHelloWorld']
```

Status: 0 Rows Returned: 1 Elapsed Time: 0.109 Execute Time: 4:01:26 AM

Drag a column header here to group by that column

colnam	colval	locale_id	cust_lvl	mls_text
opt_nam	WebServiceHelloWorld	US_ENGLISH	10	Web Service-Hello World

The Option will have to be assigned to the role as follows.



Role Maintenance

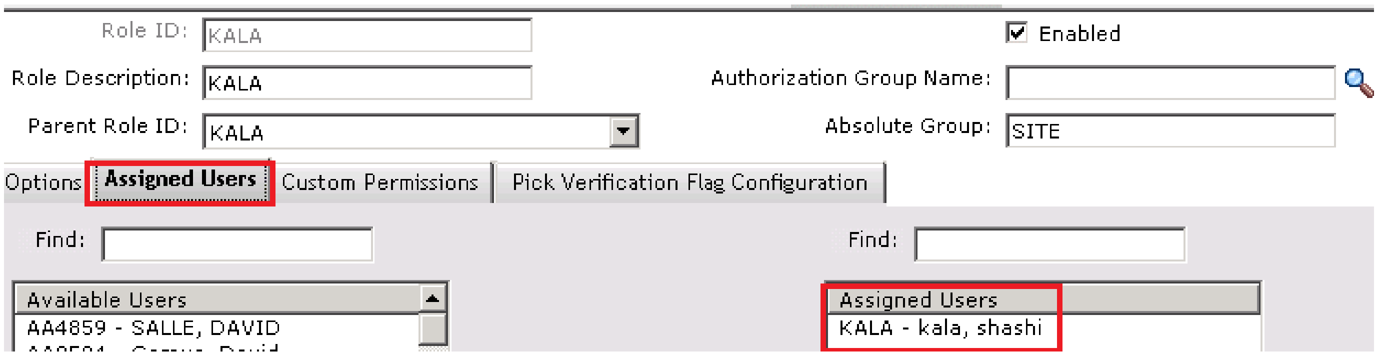
Role Description	Role ID	Enabled	Parent Role ID	Authorization Group Name	Cost Threshold	Unit Threshold	Absolute G
KALA	KALA	✓	KALA				SITE

Record 1 of 1

Role ID:	KALA
Enabled	✓
Role Description:	KALA
Authorization Group Name:	
Parent Role ID:	KALA
Absolute Group:	SITE

Options | Assigned Users | Custom Permissions | Pick Verification Flag Configuration

Available Options	Option Type	Assigned Options
MOCA Administrator	MOCA Web Services	Web Service-Hello World

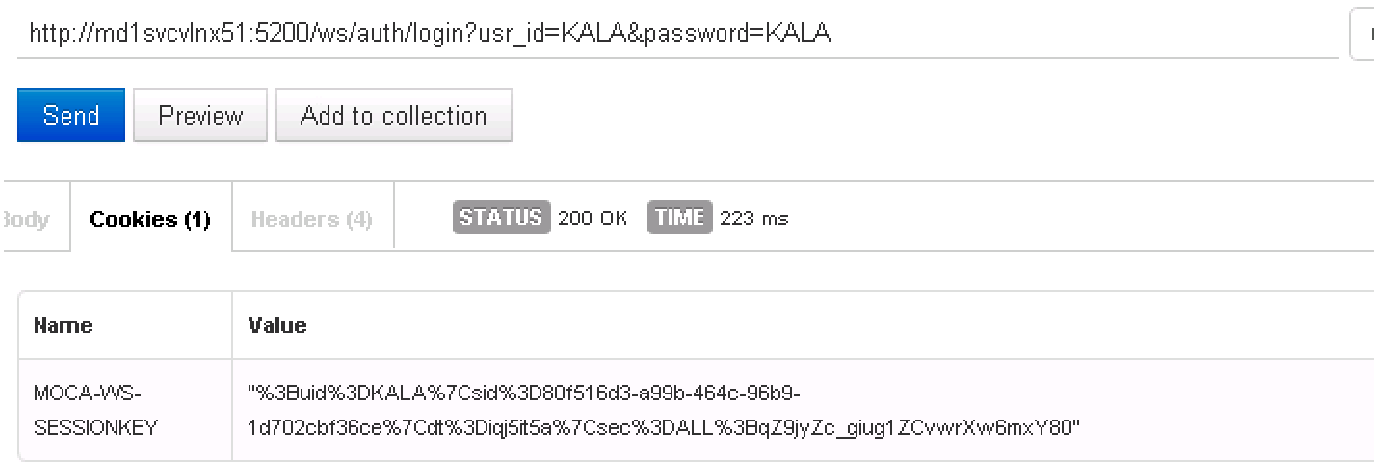


Role ID:	KALA
Enabled	✓
Role Description:	KALA
Authorization Group Name:	
Parent Role ID:	KALA
Absolute Group:	SITE

Options | Assigned Users | Custom Permissions | Pick Verification Flag Configuration

Available Users	Assigned Users
AA4859 - SALLE, DAVID	KALA - kala, shashi

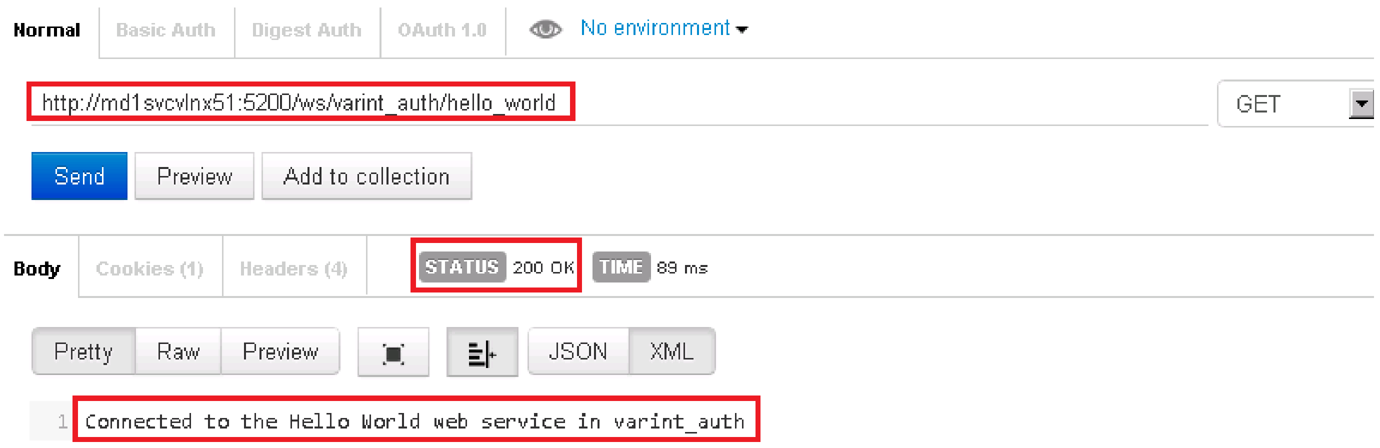
The user who has role assigned has to log on so that MOCA-WS-SESSIONKEY will be available as cookies.



http://md1svcvlnx51:5200/ws/auth/login?usr_id=KALA&password=KALA

Name	Value
MOCA-WS-SESSIONKEY	"%3Buid%3DKALA%7Csid%3D80f516d3-a99b-464c-96b9-1d702cbf36ce%7Cdt%3Diqj5it5a%7Csec%3DALL%3BqZ9jyZc_giug1ZCvwrXw6mxY80"

Now the user can access the authorized web service .



http://md1svcvlnx51:5200/ws/varint_auth/hello_world

```
Connected to the Hello World web service in varint_auth
```

Extending an Existing Web Service

Web services are not hierarchical like MOCA commands are. In MOCA, a var level command of the same name as a dcs level command is executed instead of the dcs command, and a usr level command supersedes the var level command. When using web services, each web service name must be unique. To extend an existing web service, copy the web service and add the company name to the web service name. All clients need to be updated to call the company specific web service instead of the standard web service.

Deployment

Custom web services are packaged and delivered in a rollout. The `REBUILD LES` directive runs the Java build tool `ant` to create the war files.

Revision History

Document Version	Revision Date	Author	Revision Description

1.0	09/18/2015	D Faherty	Draft
1.1	9/21/2015	D Faherty	Addressed review comments. Added error 401.
1.2	9/22/2015	D Faherty	Addressed review comments. Clarified session-key-idle-timeou t
1.3	5/5/2016	D Faherty	Added compile notes for Windows
1.4	8/12/2016	William Frederick	Updated guide to account for web service authorizations (PRFC776)

jda.
Plan to deliver™

About JDA Software, Inc.

At JDA, we're fearless leaders. We're the leading provider of end-to-end, integrated retail and supply chain planning and execution solutions for more than 4,000 customers worldwide. Our unique solutions empower our clients to achieve more by optimizing costs, increasing revenue and reducing time to value so they can always deliver on their customer promises. Using JDA, you can plan to deliver. www.jda.com

```
servicemgr –e <environment name> dump
```