Developing a plug-in connector for web service endpoints using WebSphere Cast Iron: Part 4: Implementing the DescribeObject orchestration

Raghvendra Neelekani (raneelek@in.ibm.com)
Software Engineer
IBM

04 March 2015

This five-part tutorial series shows you how to use the WebSphere Cast Iron Connector Development Kit (CDK) to quickly and easily develop plug-in connectors for web service and ReST based applications. Part 4 shows you how to implement the DescribeObject orchestration for your plug-in connector.

Introduction

Part 3 of this tutorial series showed you how to implement the ListObjects orchestration for a Salesforce.com application, including using the describeGlobal operation to list the objects from Salesforce.com using the Cast Iron WebService endpoint. Part 4 shows you how to implement the DescribeObject orchestration for your plug-in connector. The DescribeObject orchestration will be generated by the CDK wizard when any of the configured activities require dynamic discovery for their input or output schemas.

Implementing the DescribeObject orchestration

The DescribeObject orchestration is called when you select the object in the activities Browse dialog and click OK. The main purpose of this orchestration is to return the input and output schema for the selected object.

The Cast Iron CDK framework defines a schema called DescribeObjectResponse to fetch the object schema from this DescribeObject orchestration. Therefore this DescribeObject orchestration should return an XML of schema type DescribeObjectResponse. Depending on your connector design and target application APIs, you can fetch the schema from target application or get it from the wsdl file.

In Part 1, you configured two activities -- Create Objects and Get Server Timestamp -- in the CDK wizard. For the Create Objects activity, you selected Input/Output requires dynamic discovery, which means this activity requires the browsing of objects feature and dynamic discovery of
schema for the objects. Since you selected this option in the CDK wizard, the CDK framework generated this DescribeObject orchestration, in which you need to implement the logic to generate the schema.

When you open the generated DescribeObject orchestration in Cast Iron studio, you will see two preconfigured activities:

**Figure 1. Generated DescribeObject template orchestration**

- **Provide Service (DescribeObject) activity** – Entry point for Describe object operation. From this activity, you CAN Access all of the Connection fields and object types in the Map Outputs section, as shown in Figure 1. The Connection fields are the ones you configured in the CDK wizard during orchestration generation.
- **Send Reply (Send Reply) activity** – Used to return the schema back to the connector.

To fetch the complete metadata of objects from the Salesforce.com application, Salesforce.com provides a SOAP API named describeSObjects in the partner.wsdl. Therefore, implementing this template DescribeObject orchestration for Salesforce.com involves using the Cast Iron WebService Invoke Service activity to log in to Salesforce.com, invoke the describeSObjects operation, and send the result back to the connector. Here is the procedure to implement the DescribeObject orchestration for Salesforce.com:

**Configure Provide Service activity**

1. Click **Provide Service** activity in the orchestration and click **Map Outputs**.
2. Click **Copy=>**, select the variable **header** in the list of output parameters, and click **Create** to copy the connectionFields from the **header** parameter to an orchestration variable **header** that can be used by other activities.
3. Double-click the **header** variable on the To Orchestration side and rename it to **SalesforceConnectionFields**.

4. Copy the **objectType** field to a new orchestration variable to store the selected object: Click **Select Outputs** as shown in Figure 2.

5. Click **New** in the Select Outputs dialog.

6. In the Create New Variable Step 1 of 2 dialog, select **string** type under Primitive Types and click **Next**.

7. In the Create New Variable Step 2 of 2 dialog, rename the variable name from string to **objectType** and then click **Finish**.

8. Click **OK** in the Select Outputs dialog.

9. Map the **objectType** field from the left side to the newly created variable **objectType** on the right side.

**Figure 2. Creating a new orchestration variable of type string to store the object type**

Configure Salesforce Login activity

1. You need the WebService endpoint to make web service calls to Salesforce.com. If you already created the WebService endpoint for Salesforce.com in Part 1, then use that. Otherwise, create a new endpoint: In the Projects tab, right-click **Endpoints => Create Endpoint** and select **WebService**.
2. In the WebService endpoint panel, click **Browse** for the WSDL Name field and select **partner.wsdl** to load the Salesforce.com partner.wsdl to this WebService endpoint, as shown in Figure 3. This WebService endpoint represents a Salesforce.com endpoint that you can use to make calls to Salesforce.com using web services defined in the partner.wsdl.

**Figure 3. WebService Endpoint panel configured with Salesforce.com partner wsdl**

3. Rename this newly created WebService endpoint to **SalesforceEndpoint**: Right-click the newly created WebService endpoint and select **Rename**.

4. Drag and drop the **SalesforceEndpoint** endpoint to the orchestration pane between the Provide Service and Send Reply activities, and then select the **Invoke Service** activity. You can use this activity to invoke web services to a Salesforce.com application.

5. Click the newly added **Invoke Service** activity in the orchestration. An activity checklist is displayed at the bottom.

6. Go to Summary section, rename the activity to **Salesforce Login**, and press **Enter**.

7. Go to the Configure section and select the **login** operation from the list of WSDL operations to configure this activity to make a Salesforce.com login web service call.

8. Go to Map Inputs section and you will see two fields -- **username** and **password**. These fields are required for the Salesforce.com login operation.

9. The URL field is an optional parameter for this activity, so you need to enable it by adding optional parameters as shown in Figure 4. Right-click the blank space on the **To Activity** side and select **Show Optional Parameters** to display all of the optional parameters for this activity. You will see a field location to which you need to map the URL from Connection fields.
Figure 4. Enabling the optional parameters for an activity

![Image of enabling optional parameters](image)

10. Map the Salesforce Connection fields to this activity input: Click **Select Inputs**, select the `SalesforceConnectionFields` variable, and click **OK**. Map the Connection fields to the `Salesforce Login` activity as shown in Figure 5.

Figure 5. Mapping from Salesforce Connection fields to WebService Invoke Service activity input

![Image of mapping fields](image)

11. Go to the Map Outputs section, copy the parameter `body` to a new variable on the right side, and rename the variable `loginResponse`.

Configure Salesforce describe object activity

This section shows you how to configure one more Webservice activities to connect to a Salesforce.com application and fetch the metadata for the selected object, using the Salesforce describeSObjects wsdl operation.

1. Drag and drop the `SalesforceEndpoint` endpoint to the orchestration between the Salesforce Login and Send Reply activities and select the **Invoke Service** activity.
2. Click that new activity to get the check list at the bottom.
3. Go to Summary section and rename the activity `Fetch object metadata`, then press **Enter**.
4. Go to the Configure section and select the `describeSObjects` operation from the list of WSDL operations to configure this activity to make a Salesforce.com describeSObjects webservice call.
5. Go to Map Inputs section and you will see one element named sObjectType displayed under describeSObjects complex type. Map the selected object type to this sObjectType field.

6. To map the objectType field, click Select Inputs, select objectType variable, and click OK. Map the objectType variable from the left side to the objectType field on the right side.

7. Provide a login session id and server URL to this activity: Enable all the optional parameters, and you will see location and header parameters.

8. Map the Salesforce login session id and server URL to this activity input: Click Select Inputs, select loginResponse variable, and click OK. Map the login response fields to this activity as shown in Figure 6.

Figure 6. Mapping from Salesforce login response to WebService Invoke Service activity input

9. Go to the Map Outputs section, copy the parameter body to a new variable on the right side, and rename the variable describeSObjectsResponse.

Configure Apply XSLT activity

After you fetch the metadata of a selected object from a Salesforce.com application, it will be in the describeSObjectsResponse schema type defined by Salesforce.com. But the Cast Iron CDK framework expects the object schema in the form of the XML schema DescribeObjectResponse from this orchestration. Therefore you need to transform the Salesforce.com describeSObjectsResponse XML to Cast Iron defined DescribeObjectResponse XML. To transform XML from one type to another, use Cast Iron's built-in activity Apply XSLT. In this tutorial you will use the XSLT in Listing 1 below to transform the XML from describeSObjectsResponse type to DescribeObjectResponse type:

Listing 1. XSLT script to transform XML from describeSObjectsResponse to DescribeObjectResponse type

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
    xmlns:con="http://www.approuter.com/schemas/cdk/config/"
    xmlns="http://www.approuter.com/schemas/cdk/api/">
```
Developing a plug-in connector for web service endpoints using WebSphere Cast Iron: Part 4: Implementing the DescribeObject orchestration

```xml
<xsl:output indent="yes" method="xml" encoding="UTF-8"/>
<xsl:template match="/">
  <xsl:apply-templates select="child::node()"></xsl:apply-templates>
</xsl:template>
<xsl:template match="child::node()">
  <xsl:element name="DescribeObjectResponse" namespace="http://www.approuter.com/schemas/cdk/api/">
    <xsl:namespace name="xsi">http://www.w3.org/2001/XMLSchema-instance</xsl:namespace>
    <xsl:namespace name="con">http://www.approuter.com/schemas/cdk/config/</xsl:namespace>
    <xsl:attribute name="baseType" select="'sObject'"/>
    <xsl:attribute name="baseTypeNS" select="'urn:sobject.partner.soap.sforce.com'"/>
    <xsl:attribute name="typeNS" select="'urn:sobject.partner.soap.sforce.com'"/>
    <xsl:apply-templates select="*:result"/>
  </xsl:element>
</xsl:template>
<xsl:template match="*:result">
  <xsl:attribute name="name" select="*:name/text()"/>
  <xsl:attribute name="object" select="true()"/>
  <xsl:element name="object">
    <xsl:attribute name="name" select="*:name/text()"/>
    <xsl:attribute name="label" select="*:label/text()"/>
    <xsl:attribute name="minCount">1</xsl:attribute>
    <xsl:attribute name="maxCount">1</xsl:attribute>
    <xsl:attribute name="used">true</xsl:attribute>
    <xsl:apply-templates select="*:fields"/>
  </xsl:element>
</xsl:template>
<xsl:template match="*:fields">
  <xsl:if test="*:type/text() != 'location' and not(*:updateable/text() = 'false'

and :createable/text() = 'false')">
    <xsl:element name="con:field">
      <xsl:attribute name="name">
        <xsl:value-of select="*:name/text()"/>
      </xsl:attribute>
      <xsl:attribute name="label">
        <xsl:value-of select="*:label/text()"/>
      </xsl:attribute>
      <xsl:attribute name="configurable">
        <xsl:value-of select="'true'"/>
      </xsl:attribute>
      <xsl:attribute name="used">
        <xsl:value-of select="'true'"/>
      </xsl:attribute>
      <xsl:attribute name="minCount">0</xsl:attribute>
      <xsl:attribute name="maxCount">1</xsl:attribute>
      <xsl:choose>
        <xsl:when test="*:type/text() = 'picklist' or *:type/text() = 'multipicklist' or *:type/text() = 'combobox' and *:picklistValues">
          <xsl:element name="con:type">string</xsl:element>
          <xsl:attribute name="name">
            <xsl:value-of select="*:name/text()"/>
          </xsl:attribute>
          <xsl:attribute name="label">
            <xsl:value-of select="*:label/text()"/>
          </xsl:attribute>
          <xsl:attribute name="configurable">
            <xsl:value-of select="true"/>
          </xsl:attribute>
          <xsl:attribute name="used">
            <xsl:value-of select="true"/>
          </xsl:attribute>
          <xsl:attribute name="minCount">0</xsl:attribute>
          <xsl:attribute name="maxCount">1</xsl:attribute>
          <xsl:attribute name="restricted">
            <xsl:value-of select="*:restrictedPicklist/text() or *:type/text() = 'combobox'"/>
          </xsl:attribute>
          <xsl:apply-templates select="*:picklistValues"/>
        </xsl:when>
        <xsl:when test="*:type/text() = 'email' or *:type/text() = 'phone' or *:type/text() = 'url' or *:type/text() = 'percent' or *:type/text() = 'currency'">
          <xsl:element name="con:type">string</xsl:element>
        </xsl:when>
        <xsl:when test="*">
          <xsl:element name="con:format">xsl:value-of select=""/>
        </xsl:when>
      </xsl:choose>
    </xsl:element>
  </xsl:if test="true()">
</xsl:template>
```

Follow these steps to use the Cast Iron Apply XSLT activity with the XSLT script in Listing 1:

1. Save the XSLT script in Listing 1 to a file named dor.xsl.
2. Import the XSLT script file into Cast Iron studio. In the Projects tab, right-click Stylesheets, select Add Document option, browse the saved dor.xsl file, and click OK.
3. Create a new variable of schema type DescribeObjectResponse defined by the CDK framework. In the Variables tab, right-click the Variables section and select Create New Variable.
4. Select the DescribeObjectResponse element and click Next, as shown in Figure 7:

Figure 7. Creating a variable of type DescribeObjectResponse schema type

5. Click Finish.
6. Drag and drop the Apply XSLT activity to the orchestration pane between the Fetch object metadata and Send Reply activities.
7. Click Apply XSLT activity. In the Pick Stylesheet tab, click Browse.
8. In the Project Explorer dialog, select dor and click OK. You will see that the Apply XSLT activity is configured to use the provided dor XSLT script, as shown in Figure 8:
9. Click **Set Input & Output**.
10. In the input section, browse and select the `describeSObjectsResponse` variable.
11. For the output section, browse and select the `DescribeObjectResponse` variable as shown in Figure 9:

**Figure 9. Apply XSLT activity mapping**
Configure Send Reply activity

This section shows you how to configure the Send Reply activity to send the Describe object response XML back to the connector. The describe object response XML is in the DescribeObjectResponse variable, and you need to use this variable in the Send Reply activity, and map it to its input as shown in Figure 10 below.

1. Click the Send Reply activity. The Checklist for this activity opens at the bottom.
2. Click Map Inputs.
3. Click Select Inputs.
4. Select the DescribeObjectResponse variable from the list of variables and click OK.
5. From the Functions tab, drag and drop the CopyOf function to the middle of the Map Inputs pane.
6. Do the mapping from the top level element DescribeObjectResponse on the left side to the dropped CopyOf function, as shown in Figure 10.
7. Do the mapping from the dropped CopyOf function to the right side the DescribeObjectResponse element, as shown in Figure 10:

Figure 10. Send Reply activity mapping

8. Right-click that function and select Apply Function Graph.

You have now configured a valid DescribeObject orchestration. As a best practice, add a Try-Catch block to the entire orchestration to handle error conditions. In this example, you will configure one more Send Reply activity to send an empty DescribeObjectResponse XML in the catch block.

Configure Try-Catch block

1. Right-click the initial orchestration image and select Add Catch Branch to add a catch block for the entire orchestration.
2. Drag and drop another Send Reply activity from SalesforceEndpoint to the CatchAll block. Rename this activity Send Empty Reply.
3. Go to the Map Inputs section of this Send Empty Reply activity.
4. Create a variable of schema type DescribeObjectResponse and provide empty DescribeObjectResponse XML, as shown in Figure 11 and provided in Listing 2.
   a. Click the Variables tab and in the Variables section, right-click and select Create New variable.
   b. Select the DescribeObjectResponse element from the cdkconnector wsdl and click Next.
   c. Rename the variable name from DescribeObjectResponse_1 to emptyDescribeObjectResponse, and then click Finish.
   d. Click the emptyDescribeObjectResponse variable, click Default value in the Properties section, and provide the default value (empty DescribeObjectResponse XML) in the Default Value dialog, as shown in Figure 11.

Listing 2. Empty DescribeObjectResponse XML

```xml
<DescribeObjectResponse xmlns="http://www.approuter.com/schemas/cdk/api/">
</DescribeObjectResponse>
```

Figure 11. Empty DescribeObjectResponse variable

5. Map the emptyDescribeObjectResponse XML variable in the Send Empty Reply activity: Click Send Empty Reply activity and click Map Inputs checklist.
6. Click Select Inputs and select the emptyDescribeObjectResponse variable.
7. Do the mapping from the `emptyDescribeObjectResponse` variable to `body`. At the end, you will see the complete orchestration as shown in Figure 12:

**Figure 12. Complete DescribeObject orchestration**

Verify DescribeObject orchestration

You can verify this DescribeObject orchestration by using the Invoke Service utility.

1. Go to the Orchestration menu and click **Invoke Service**, as shown in Figure 13. The orchestration starts and the Invoke Service window opens.

**Figure 13. Selecting the Invoke Service utility**

2. Provide your Salesforce.com credentials in the UserName, Password, URL, and objectType fields and then click **Execute** to invoke the DescribeObject orchestration and fetch the result.

3. Click **Show Response** on the right side. The response from the orchestration is displayed on the right side window, as shown in Figure 14:
To verify negative cases, provide invalid credentials and note the response from the orchestration.

**Conclusion**

Part 4 showed you how to implement the plug-in connectors DescribeObject orchestration with the example of the Salesforce.com describeSObjects operation. In Part 5, you will learn how to build activity-based orchestrations for the connector. When you created the connector, you configured two activities -- Create Objects and Get Server Timestamp in the CDK wizard. In Part 5 you will implement these two orchestrations.

**Acknowledgments**

The author would like to thank Sarath K. Ambadas and Srihari Na Kumar of IBM India for reviewing and providing valuable suggestions for this tutorial.
Resources

- **WebSphere Cast Iron resources**
  - **WebSphere Cast Iron Cloud Integration Knowledge Center**
    A single portal to all WebSphere Cast Iron Cloud Integration documentation, with conceptual, task, and reference information on installing, configuring, and using WebSphere Cast Iron.
  - **WebSphere Cast Iron Cloud Integration product library**
    Product announcements, case studies, white papers, and more.
  - **WebSphere Cast Iron Cloud Integration product page**
    Product descriptions, product news, training information, support information, and more.
  - **WebSphere Cast Iron Cloud Integration support**
    A portal for support problems and their solutions, plus downloads, fixes, problem tracking, and more.
  - **WebSphere Cast Iron Cloud Integration Community forums**
    Get answers to your technical questions and share your expertise with other WebSphere Cast Iron users.
  - **IBM Redbook: Getting started with WebSphere Cast Iron Cloud Integration**
    Detailed introduction to the development and administrative interfaces for WebSphere Cast Iron.
  - **Salesforce.com: SOAP API Developer's Guide**
    Salesforce provides programmatic access to your organization's information using simple, powerful, and secure APIs.
  - **WebSphere Cast Iron Connector Development Kit (CDK) Developer's Guide**
    For developers who want to use the WebSphere Cast Iron Studio development platform to develop and deploy plug-in connectors that extract data from application systems or other data sources.
  - **Developing a starter activity for plug-in connectors using the WebSphere Cast Iron Studio CDK**
    This developerWorks article shows you how to use the WebSphere Cast Iron CDK to develop a plug-in connector that has a starter activity, deploy the plug-in, and test it locally on WebSphere Cast Iron Studio.
  - **Hints and pointers for connector development in WebSphere Cast Iron Studio**
    This developerWorks article shows you how to easily configure the base orchestrations for connectors developed with the WebSphere Cast Iron Studio CDK wizard. Includes tips on password and fault handling.

- **WebSphere resources**
  - **developerWorks WebSphere**
    Technical resources for developers who use WebSphere and related middleware products. Includes downloads, how-to information, support resources, and a free technical library of tutorials, best practices, IBM Redbooks, and product manuals.
  - **Most popular WebSphere trial downloads**
    No-charge trial downloads for key WebSphere products.
  - **WebSphere forums and community**
Product-specific forums where you can get answers to your technical questions and share your expertise with other WebSphere users.

- **WebSphere demos**
  Download and watch these self-running demos, and learn how WebSphere products can provide business advantage for your company.

- **WebSphere-related tutorials on developerWorks**
  Nearly 3000 tutorials on WebSphere and related technologies by top practitioners and consultants inside and outside IBM. Search for what you need.

- **WebSphere-related books from IBM Press**
  Convenient online ordering through Barnes & Noble.

- **WebSphere-related events**
  Conferences, trade shows, Webcasts, and other events around the world.

- **developerWorks resources**
  - **Trial downloads for IBM software products**
    No-charge trial downloads for selected IBM® DB2®, Lotus®, Rational®, Tivoli®, and WebSphere® products.
  - **developerWorks blogs**
    Join a conversation with developerWorks users and authors, and IBM developers.
  - **Technical events for developers**
    Conferences, seminars, and technical events to deepen your technical expertise and help you succeed in your software projects.
  - **developerWorks podcasts**
    Listen to interesting and offbeat interviews and discussions with software innovators.
  - **developerWorks on Twitter**
    Check out recent Twitter messages and URLs.
  - **IBM Education Assistant**
    A collection of multimedia educational modules that will help you better understand IBM software products and use them more effectively to meet your business requirements.
About the author

Raghvendra Neelekani

Raghvendra Neelekani is a Staff Software Engineer for IBM in India. He has been working for IBM since 2009, and has extensive experience with Java and Web 2.0 technologies. He works on the WebSphere Cast Iron Development Team and has developed many Cast Iron connectors. You can contact Raghavendra at raneelek@in.ibm.com.

© Copyright IBM Corporation 2015
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)