Developing a plug-in connector for web service endpoints using WebSphere Cast Iron: Part 5: Implementing connector activity orchestrations

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 5 shows you how to implement the orchestrations for your plug-in connector activities.

View more content in this series

Introduction

In Parts 1-4 of this tutorial series, you have learned how to implement the TestConnection, ListObjects, and DescribeObject orchestrations for the Salesforce.com application, and you have used Salesforce.com web services to perform corresponding operations using a Cast Iron WebService endpoint. Part 5 shows you how to implement the orchestrations for your plug-in connector activities. These orchestrations were generated by the CDK wizard when you created the connectors, and each activity orchestration will be exposed as an activity for your connector.

Activity orchestrations

Activity orchestrations are generated based on how you configured them in the CDK wizard when creating the connectors. The number of activity orchestrations is equal to the number of activities you configured. Each orchestration is exposed as a web service to the Cast Iron CDK framework, so that when you run the activity, the corresponding orchestration is invoked. In Part 1, you configured two activities:

1. Create Objects -- To create objects in the Salesforce.com application
2. Get Server Timestamp – To get the Salesforce.com server current timestamp information.

Part 5 shows you how to implement the above two orchestrations using Salesforce.com web services.
Implementing the Create Objects orchestration

The Create Objects orchestration is called by the Cast Iron runtime engine when you execute the Create Objects activity. The main purpose of this orchestration is to connect to the Salesforce.com application and create a new record based on incoming object type and data.

When you open the generated Create_Objects orchestration in Cast Iron Studio, you see two preconfigured activities:

**Figure 1. Generated Create_Objects template orchestration**

- **Provide Service (Create_Objects) activity** – Entry point for create operation for any object type. In this activity, you will get access to all connection fields, object type, and input data 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, and the object type and complete input data are in the parameter body.

- **Send Reply (Send Reply) activity** – Used to return the response of this orchestration back to the CDK framework. If you look at the schema in the Map Inputs section of this activity, you will see a createResponse element as defined in the Salesforce.com partner wsdl, because you configured the createResponse schema as the output structure for this Create Objects activity in the CDK wizard.

To programmatically create a record of any object type in a Salesforce.com application, Salesforce.com provides a SOAP API named Create in the partner.wsdl. Therefore, in this tutorial, you will implement a Create_Objects orchestration by using the Cast Iron WebService Invoke Service activities to log in to Salesforce.com, invoke the Create operation, and send the result back to the CDK framework. Here are the steps to implement the Create_Objects 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 the orchestration variable header, which can be used by other activities.
3. Double-click the **header** variable on the To Orchestration side and rename it **SalesforceConnectionFields**.
4. Copy the input data sObjects XML to a new orchestration variable to store the input data: Click **Select Outputs**, as shown in Figure 2:

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

5. Click **New** in the Select Outputs dialog.
6. In the Create new variable Step 1 of 2 dialog, select the **create** element under partner and click **Next**.
7. In the Create new variable Step 2 of 2 dialog, rename the variable name from string to **inputSObjectsData** and then click **Finish**.
8. Click **OK** in the Select Outputs dialog.
9. Use the CopyOf function to map the parameter body to the variable inputSObjectsData, as shown in Figure 3. Now you can use the variable inputSObjectsData in later activities to access the complete input data.
10. Right-click the dropped CopyOf function and select Apply Function Graph.

Configure Password Decoding activity

The password element in the connection fields contains the Salesforce.com password, either in plain text or encoded. You need to decode it if it is encoded, or use it as it is. Here are the steps to configure a Map Variables activity to decode the password and store it in a separate variable. Then you can use an If...Then activity to check whether the decoded password is correct.

1. Drag and drop the Map Variables activity to the orchestration between the Provide Service and Send Reply activities.
2. Click Select Inputs.
3. Select the SalesforceConnectionFields variable and click OK.
4. Use the Decode Password Configuration Property function to decode the password to a new string variable decodedPassword, as shown in Figure 4. Right-click the dropped function and select Apply Function Graph.
The Decode Password Configuration Property function will decode the password if the incoming password is encoded. Otherwise this function returns an empty string.

1. Use the If...Then activity to check if the decoded password has the correct value and assign it back to the connection fields XML. Drag and drop the If...Then activity to the orchestration pane between the Map Variables and Send Reply activities.
2. Click If part. The Logic if pane is displayed in the bottom.
3. In the left hand Expression, select the `decodedPassword` variable, as shown in Figure 5, and click OK:

**Figure 5. Selecting decodedPassword variable in the If block**
4. Select the \(!=\) operator in the Op column.
5. In the right hand expression, provide an empty string by entering two single quotes (\"\"). With this configuration, the execution will go to the If block if the decoded password is not an empty string and has a valid value.
6. Drag and drop one more Map Variables activity inside the If block.
7. Click Select Inputs and select the decodedPassword variable.
8. Click again on Select Inputs and select the SalesforceConnectionFields variable.
9. Click Select Outputs and select the SalesforceConnectionFields variable.
10. Do the mapping from decodedPassword to the password element under ConnectionFields, as shown in Figure 6.
11. Do the mapping from UserName to UserName and from Password to Password, as shown in Figure 6:

   Figure 6. Decode password mapping

12. You can delete the Else block: Right-click Else block and select Delete.

### 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 Endpoint => 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 7. 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.
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 onto the orchestration after the **IfThen** block and select the **Invoke Service** activity to invoke web services to the Salesforce.com application.

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

6. Go to the Summary section, rename the activity `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 the 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 will be an optional parameter for this activity, so you need to enable it by adding optional parameters. Right-click the blank space in the **To Activity** side and select **Show Optional Parameters**. All optional parameters of this activity are displayed, including the **Location** field, to which you need to map the URL from the Connection fields.

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 8:
Developing a plug-in connector for web service endpoints using WebSphere Cast Iron: Part 5: Implementing connector activity orchestrations

Figure 8. Mapping from Salesforce connection fields to WebService Invoke Service activity input

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 create objects activity**

This section shows you how to configure one more Webservice activities to connect to the Salesforce.com application and create a new record, using the Salesforce.com create SOAP 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 checklist at the bottom.
3. Go to the Summary section and rename the activity to **create objects**, then press **Enter**.
4. Go to the Configure section and select **create** operation from the list of WSDL operations, to configure this activity to make a Salesforce.com **create** web service call.
5. Go to the Map Inputs section. You will see one recurring element named **sObjects** displayed under the create complex type. Map the incoming **sObjects** data to the create element.
6. To map the input sObjects data, click **Select Inputs**, select the **inputSObjectsData** variable and click **OK**.
7. Use the **CopyOf** function to map the **create** element from the **inputSObjectsData** variable to the right side create element, as shown in Figure 9.
8. Provide a login session id and server URL for this activity: Enable all optional parameters and you will see the location and header parameters.
9. Map the Salesforce.com login session id and server URL to this activity input: Click **Select inputs**, select the **loginResponse** variable, and then click **OK**. Map the login response fields to this activity, as shown in Figure 9.
10. Go to the Map Outputs section and copy the **body** to a new variable on the right side and rename the variable **createResponse**.

**Configure Send Reply activity**

This section shows you how to configure the Send Reply activity to send the Create operation response XML back to the CDK framework. The Create operation response XML is in the `createResponse` variable, and you need to use this variable in the Send Reply activity and map it to its input.

1. Click the **Send Reply** activity. You will see the Checklist for this activity in the bottom.
2. Click **Map Inputs**.
3. Click **Select Inputs**.
4. Select the `createResponse` variable from the list of variables, and then click **OK**.
5. Do the mapping from the `createResponse` element on the left side to the `createResponse` element on the right side.

You have now configured a valid Create_Objects orchestration, which can connect to Salesforce.com application and create a new record on the server. As a best practice, add a Try-Catch block to the entire orchestration to handle error conditions. Configure one more Send Reply activity to send the error messages back to the CDK framework.

**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 Error Reply**.
3. Go to the **Map Inputs** section of this Send Error Reply activity.
4. Click **Select Inputs** and select the **faultInfo** variable, which is an XML variable autogenerated by Cast Iron Studio that contains information about failure of the orchestration.
5. Map the **message** element from **FaultInfo** to the **message** element under createResponse.
6. Provide a default value of **false** for element success under the createResponse element. At the end, you will see the complete orchestration, as shown in Figures 10 and 11:

**Figure 10. Part 1 of complete Create_Objects orchestration**

**Figure 11. Part 2 of complete Create_Objects orchestration**

**Get Server Timestamp orchestration**

The purpose of describing the Get Server Timestamp orchestration in this tutorial is to help you understand how to configure an activity that does not require any input or output data from the end user. The purpose of this orchestration is to connect to the Salesforce.com application, get the current server timestamp, and return it back to the framework. This orchestration is called by the Cast Iron runtime engine when you execute the Get Server Timestamp activity.

**Analysis for Get Server Timestamp orchestration**

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

- **Provide Service (Get_Server_Timestamp)** – This Provide Service activity is the entry point when you invoke the Get Server Timestamp activity. This activity gives you access to all of the Connection fields in the Map Outputs section. You will see one more element, getServerTimestamp, in the parameter body. You can ignore this parameter, since user will not provide any value for it. You can copy the Connection fields to an orchestration variable and use them in the later activities.

- **Send Reply (Send Reply)** – The Send Reply activity is used to return the timestamp response back to the CDK framework. If you look at the schema in the Map Inputs section...
of this activity, you will see the `getServerTimestampResponse` element as defined in the Salesforce.com partner wsdl, because you configured the `getServerTimestampResponse` schema as output structure for this Get Server Timestamp activity in the CDK wizard when you created the connector.

In order to programmatically get the timestamp information from a Salesforce.com application, Salesforce.com provides a SOAP API `getServerTimestamp` that you can find in the partner.wsdl. Hence you can implement this orchestration by using the Cast Iron WebService Invoke Service activities to log in to Salesforce.com, invoke the `getServerTimestamp` operation, and send the result back to the CDK framework.

**Implementing the Get Server Timestamp orchestration**

In order to implement the Get Server Timestamp orchestration, you can follow the implementation of the Create Objects orchestration to configure most of the activities. Here is the The Get Server Timestamp orchestration activities flow:

1. Copy the Connection fields from the Provide Service activity to a new orchestration variable. See the section Configure Provide Service activity.
2. Configure the password decoding activity to get the valid password value. See the section Configure Password Decoding activity.
3. Use the WebService Invoke activity to log in to Salesforce.com using the Connection fields. See the section Configure Salesforce Login activity.
4. Use the WebService Invoke activity to invoke the Salesforce.com `getServerTimestamp` web service, as shown in Figure 12:

**Figure 12. Mapping of `getServerTimestamp` activity**

```plaintext
| Invoke Service (6) | Salesforce.com Login | Invoke Service (20) | `getServerTimestamp` | Send Reply (5) |
```

<table>
<thead>
<tr>
<th>Map Inputs</th>
</tr>
</thead>
</table>

From Orchestration: `loginResponse`

To Activity: `getServerTimestamp`
5. Map the response of the getServerTimestamp activity to the Map Inputs of the Send Reply activity. The complete orchestration flow is shown in Figures 13 and 14:

**Figure 13. Part 1 of complete Get_Server_Timestamp orchestration**

![Figure 13. Part 1 of complete Get_Server_Timestamp orchestration](image)

**Figure 14. Part 2 of complete Get_Server_Timestamp orchestration**

![Figure 14. Part 2 of complete Get_Server_Timestamp orchestration](image)

You have now implemented both of the orchestrations and completed project development for the plug-in connector. To validate the completeness of the project, click **Validate Project** as shown in Figure 15. A window should open with the message **The Studio project is valid**. If there are any errors in the project that you need to resolve, a window will open and list them.

**Figure 15. Verifying the studio project**

![Figure 15. Verifying the studio project](image)

**Publishing the plug-in connector to repository**

After you develop a valid plug-in connector project, you can publish it to a local repository. The location of the local repository on your machine will be `C:/Users/<username>/.castiron/connector-repository`. You can then use the connector to develop business integration orchestrations. To publish the connector to a local repository:

1. In the menu item, select **File => Publish Connector To Local Repository**. The plug-in connector gets published and the SalesForceDemoConnector is available for use.
2. A message window opens after the connector project is successfully published. Click **OK**, and then close Cast Iron Studio.

**Testing the SalesForceDemoConnector plug-in connector**

You can use the SalesForceDemoConnector connector in Cast Iron Studio to build business integration orchestrations. To test the SalesForceDemoConnector connector and its activities, perform the following steps:

1. Open Cast Iron Studio and create a new project called `TestPluginConnector`:
   1. Select **File => New Project**.
   2. Specify `TestPluginConnector` as the Project Name:

   ![Create New Project](image)

   **Figure 16. Create New Project**

   3. Click **OK**.

2. Create the `SalesForceDemoConnector` endpoint:
   1. In the Project tab, right-click **Endpoints**, and select **Create Endpoint => SalesForceDemoConnector**, as shown in Figure 17. This plug-in connector is the one you developed in this tutorial:
3. Fill in the SalesForceDemoConnector endpoint connection parameters and test the connection:

1. In the Projects tab, double-click **SalesForceDemoConnector** in the Endpoints section.
2. The SalesForceDemoConnector endpoint panel opens showing three connector parameters: UserName, Password, and URL. You configured these parameters in the Cast Iron CDK wizard when creating the connector. Enter your Salesforce.com credentials in the UserName, Password, and URL fields to connect to Salesforce.com:

**Figure 18. SalesForceDemoConnector endpoint panel with connection details**
3. Click **Test Connection** to invoke the TestConnection orchestration in the SalesForceDemoConnector connector project that you developed earlier. A message shows the status of the test connection.

4. Configure the Create Objects activity. When you expand the **SalesForceDemoConnector** folder under the Activities tab as shown in Figure 19, you will see the two activities that you configured in the CDK wizard when creating the connector.

5. Drag and drop the **Create Objects** activity to the orchestration from the Activities tab, as shown in Figure 19:

   **Figure 19. Create Objects activity configuration**

6. Click the **Create Objects** activity that was added to the orchestration. Add the **SalesForceDemoConnector** endpoint in the Pick Endpoint check list.

7. Click **Configure** and then **Browse**. The CDK framework invokes the ListObjects orchestration that you developed in Part 2 and fetches the object list, which is displayed in a window:
8. Select **Account** object type and click **OK**. The CDK framework invokes the DescribeObject orchestration and passes the object type you selected (Account in this case) and the and connection fields. After the CDK framework gets the schema for the Account object type, it populates the schema in the Map Inputs section of this activity, as shown in Figure 21:

**Figure 21. Account object type schema populated in the Map Inputs**
9. In the Map Inputs section, right-click the **Name** field under the {Account}sObjects element and select **Define Default value**. A Define Default Value window opens.

10. Enter the default value **TestingConnectorForMe**, as shown in Figure 22. It will populate only the **Name** field to create a record of Account object type in the Salesforce.com application.

**Figure 22. Providing default value**

11. After you browse the object type and provide input data to the activity, you can verify the activity to test it: Right-click the **Create Objects** activity in the orchestration and Click **Verify Activity**:

**Figure 23. Verifying the activity**

12. After you verify the activity, Cast Iron studio starts the orchestration and invokes the Create_Objects orchestration that you developed in this tutorial. You can see the result of the activity test in the Verify pane:
Figure 24. Activity result in Verify pane

13. You can test the new Account object created in the Salesforce.com application as shown in Figure 25. Log in to https://login.salesforce.com and select the **Account** tab. You will see new Account record created with the name **TestingConnectorForMe**.

Figure 25. New Account created in Salesforce.com application
You can also verify the **Get Server Timestamp** activity by dragging it to the orchestration pane and verifying it.

## Conclusion

This five-part tutorial series showed you how to:

- Understand the Cast Iron CDK framework and plug-in connector development.
- Use the Cast Iron CDK Wizard to generate artifacts to develop a plug-in connector for web service based endpoints such as Salesforce.com.
- Develop a plug-in connector with two activities.
- Publish and test the plug-in connector in Cast Iron Studio.

## 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

Raghavendra 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/)