

Using the REST Adapter

Objectives

After completing this lesson, you should be able to:

- Identify the format of REST queries
- List the differences between REST and SOAP
- Create and test a REST binding in a composite application



Agenda

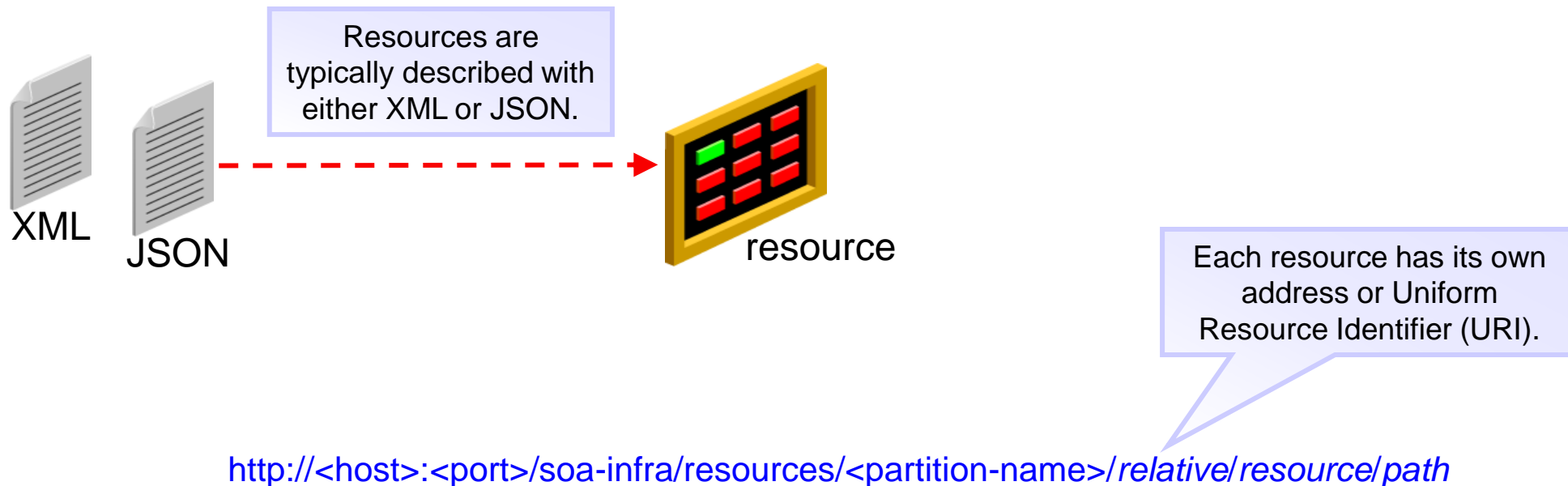
- REST: Overview
- Using REST bindings in Oracle SOA Suite



REST: Overview

REST

- Stands for *Representational State Transfer*
- Provides an alternative to using SOAP-based web services
- Incorporates the concept of *resources*. A resource is similar to an object instance in an object-oriented programming language. Resources have data associated with them.



REST Queries

Example REST query:
`http://soa12c.example.com/items/123`

Action	Method	URI
Get all the items.	GET	/items
Get a single item.	GET	/items/id
Create a new item.	POST	/items
Edit an item.	PUT	/items/id
Delete an item.	DELETE	/items/id

Comparing REST and SOAP: Two Ways to Access Web Services

REST	SOAP
Is an architectural style that leverages web standards	Is a formal standard for message exchange
Uses HTTP	Is protocol independent
Permits many data formats	Uses XML
Uses URI and HTTP verbs to access <i>resources</i> (data)	Uses a WSDL document to access <i>operations</i> (business logic)
Is stateless	WS* standards provide support for stateful transactions.

Example Use Cases for REST and SOAP

REST

- Operations with limited bandwidth and resources
 - Mobile
 - Series of short, chatty conversations
- Stateless operations

SOAP

- Operations that require contextual information and conversational state management
- Asynchronous operations
- Operations that require high levels of security and reliability

Agenda

- REST: Overview
- Using REST bindings in Oracle SOA Suite

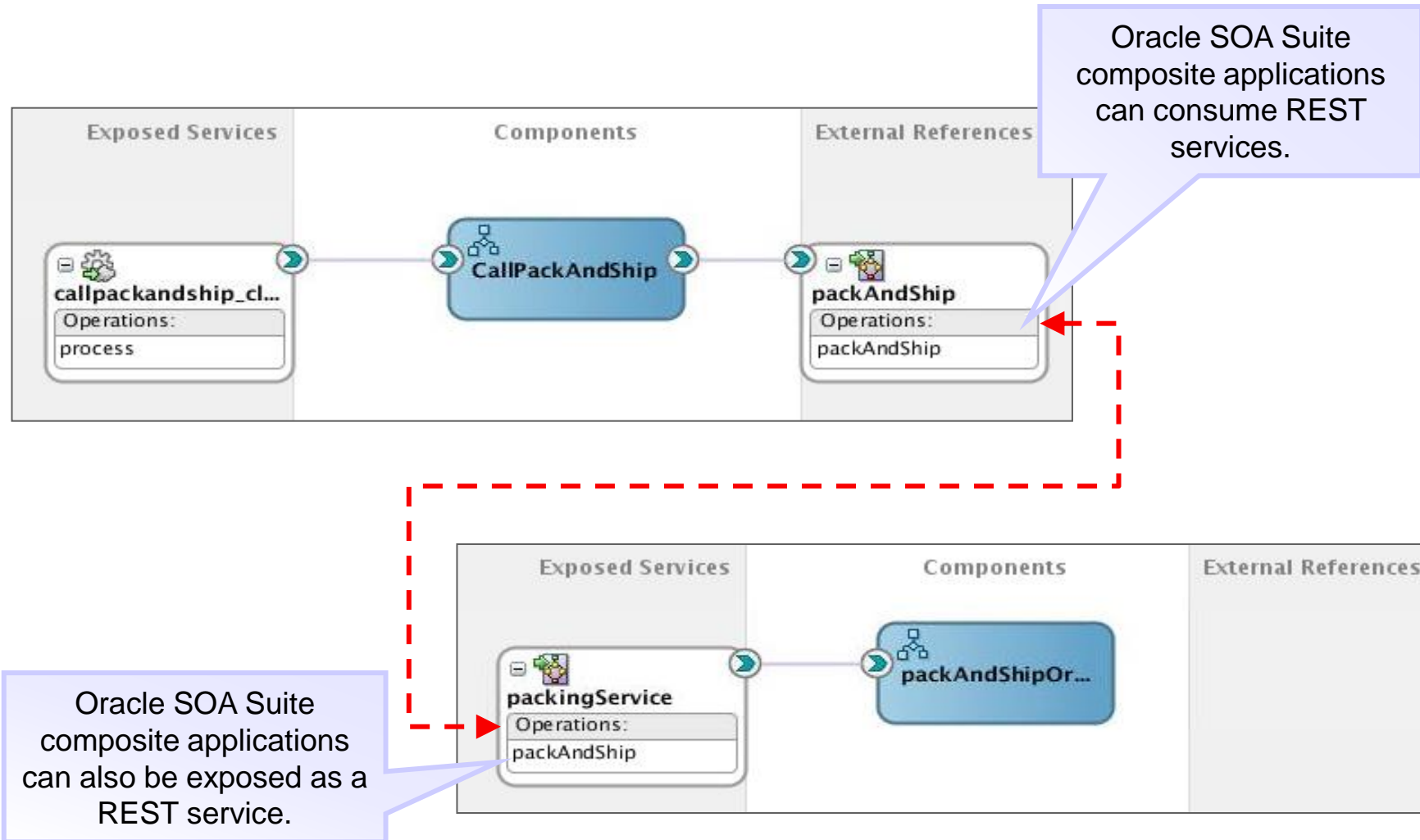


Supported Features in Oracle SOA Suite

Oracle SOA Suite provides the following REST support:

- Enabling REST support in new or existing services
- Integration with external REST APIs
- Orchestration of a set of RESTful state transitions
- Support for XML and JSON
- Generation of sample URI for REST service operations
- Support for WADL services
- Support for REST security with the Oracle Web Service Manager (OWSM) policy

REST Adapter



Configure the REST Adapter

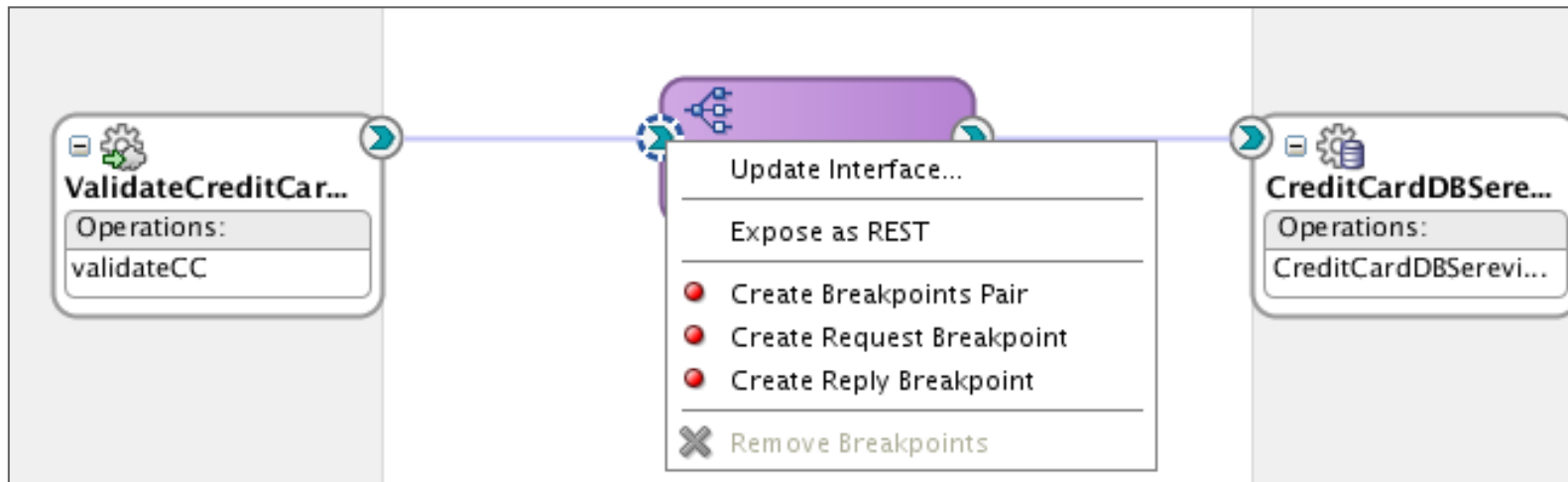
- Configuring the REST Adapter means choosing:
 - Will it use XML or JSON within the composite
 - Note: The Native Format Builder can easily generate the needed XSDs to support JSON to XML transformations internally.
 - What will be the URI, Resources, operations and parameters it exposes
- In some cases, the adapter is initially configured for you
- You can choose to manually define the Adapter or use a configuration shortcut based on a WADL or existing WSDL

REST Adapter Scenarios

- Add a REST interface to an existing composite SOAP interface implemented by a mediator or BPEL process.
- Create a custom REST Exposed Service Interface for a composite.
- Create a new REST External Reference for an external service or component.

Scenario 1: Expose an Existing SOAP Composite with a REST Interface

- The easiest scenario is exposing an existing SOAP composite with a REST interface.
- Choose the component that implements the interface: mediator or BPEL and click to Expose as REST.
- You are simply exposing the same operations supported by the WSDL, but as a REST interface.



Configure the Adapter

- If you expose an existing SOAP-based composite, its WSDL and XSD schemas are used to translate incoming payloads into XML.
- You work with this XML in the same way in the mediator and BPEL - using XPath.

REST Binding Configuration Wizard - Step 1 of 2

REST Binding

Name the Adapter

Name: RestService

Type: Service

☒ Service will invoke components using WSDL interfaces
If checked, this binding will map the REST resources/verbs to internal WSDL operations and XML schemas, with incoming payloads being translated to XML.

☐ Enforce XMLSchema Ordering

Defining Resources

- When exposing an existing SOAP-based composite, the operation(s) exposed in the WSDL are already defined.
- You define the Resource Path for each operation.
- You choose what HTTP Verb is called when the service's operation is invoked.

REST Binding Configuration Wizard - Step 2 of 2

Resources

Configuration Shortcut: +

Description: RestService

Resources: + - x

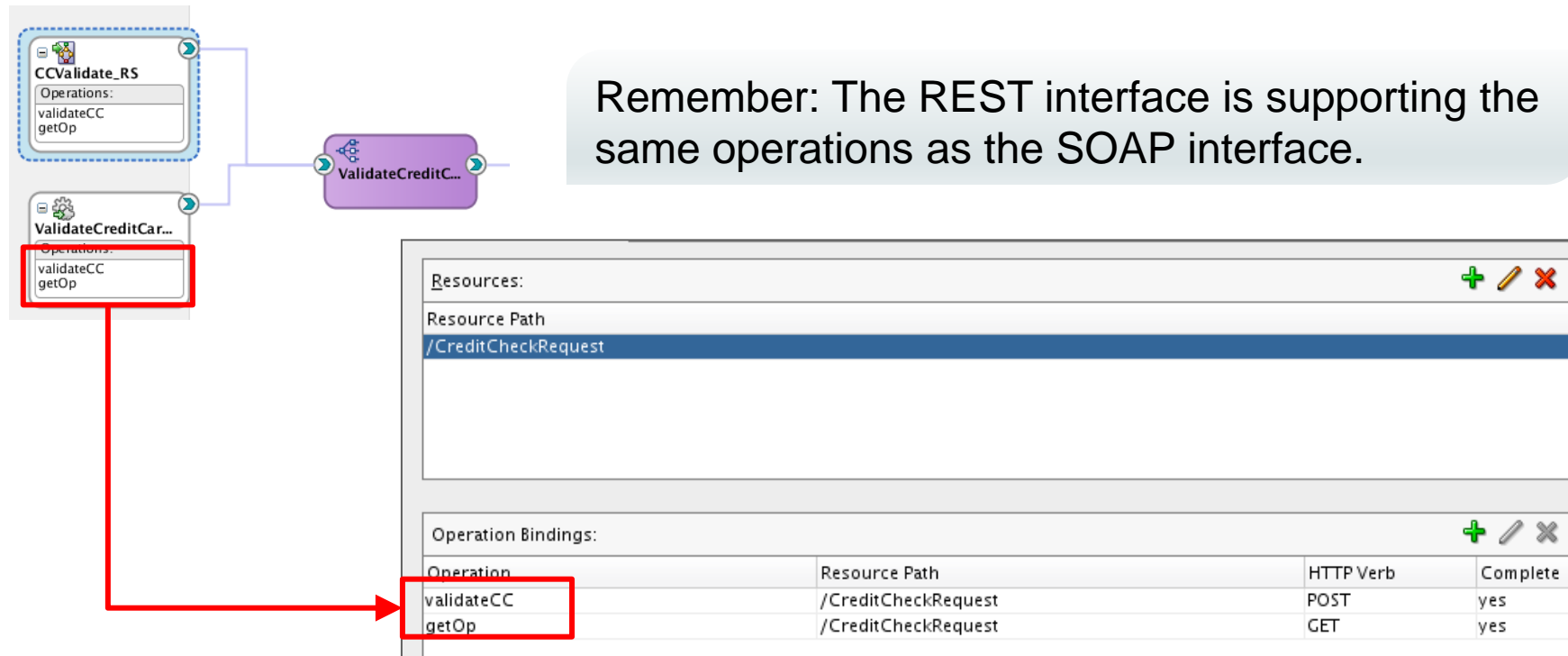
Resource Path
/

Operation Bindings: + - x

Operation	Resource Path	HTTP Verb	Complete
validateCC	/	<<Not Configure...	no

Supporting Additional Operations in the REST Interface

- In this example, the WSDL was extended to add an additional operation and the Resource Path was renamed.



Define Operation Binding

- For SOAP-based composites, the schema, request parameters, style, type and expression are inferred from the WSDL.
- Choose the HTTP Verb for the operation.

REST Operation Binding

Operation:

Resource:

HTTP Verb:

Description:

Schema URL:

Element:

URI Parameters:

Parameter	Description	Style	Type	Default Value	Expression
amount		query	decimal		\$msg.part1/inp1:amount
CCNumber		query	string		\$msg.part1/inp1:CCNu...

Define the Response

- Choose what media(s) the payload should be.
- You can preview a sample of what the payload will look like.

The screenshot shows the 'REST Operation Binding' dialog box. The 'Operation' field is set to 'validateCC' and the 'Resource' is '/'. The 'HTTP Verb' is 'GET'. The 'Description' field is empty. Below these fields are two tabs: 'Request' and 'Response'. The 'Response' tab is selected. In the 'Response' tab, the 'Schema URL' is '../Schemas/creditcheck.xsd' and the 'Element' is 'CreditCheckResponse'. The 'Payload' section has checkboxes for 'JSON' (checked), 'XML' (checked), 'URL-encoded' (unchecked), and 'No payload' (unchecked). There is a 'Generate Sample Payloads' button next to the payload checkboxes. The 'HTTP Statuses' field is set to '200'. A 'Fault Bindings' section is at the bottom with a '+' icon.

Operation: validateCC

Resource: /

HTTP Verb: GET

Description:

Schema inferred from the WSDL

Request Response

Schema URL: ../Schemas/creditcheck.xsd

Element: CreditCheckResponse

Payload: ☒ JSON ☒ XML ☐ URL-encoded ☐ No payload

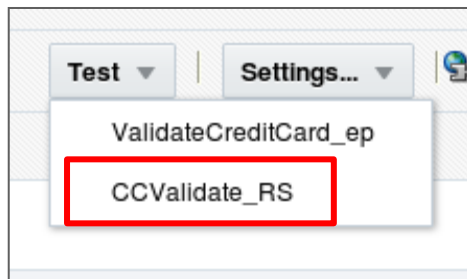
Generate Sample Payloads

HTTP Statuses: 200

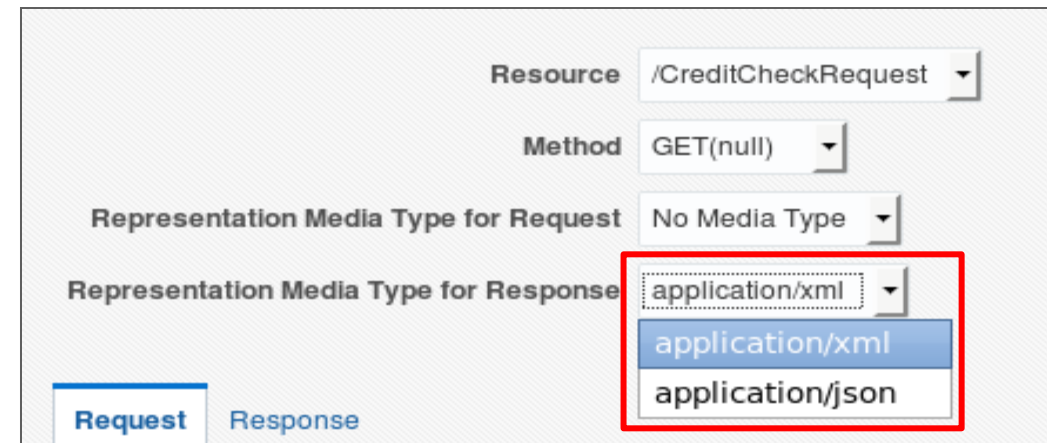
Fault Bindings:

Testing the New REST Interface

- When you test the composite in the Fusion Middleware Control, you will see additional choices, one for each new REST endpoint.

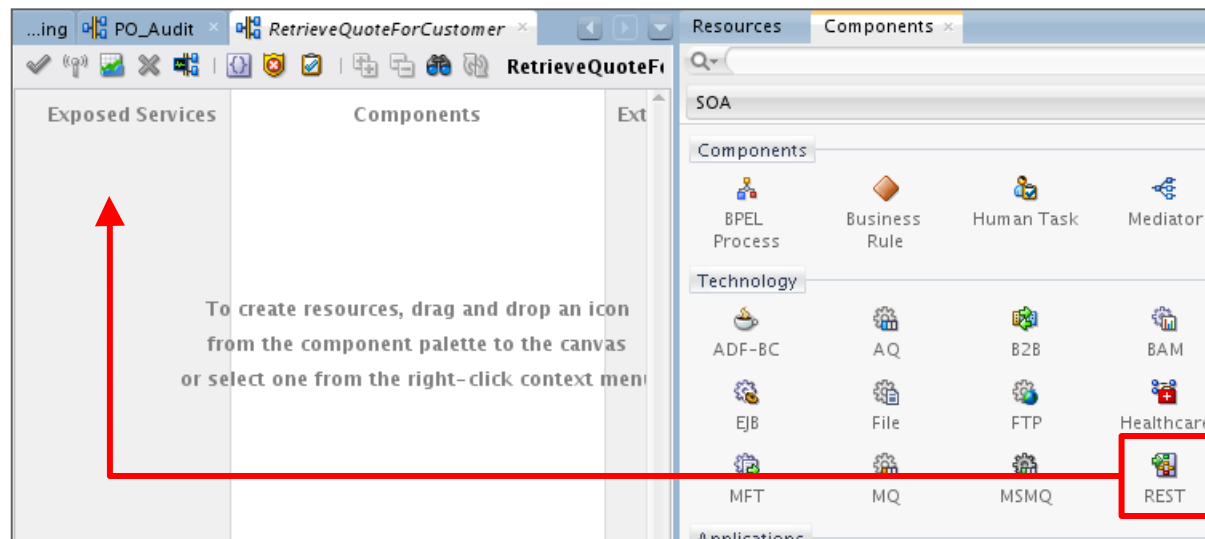


The test page will show the resource path name and choices for the different media types it can receive and return.



Scenario 2: Create a Custom REST Exposed Service Interface for a Composite

- In this scenario, you are create a custom REST interface to support the Resources and operations your service needs.

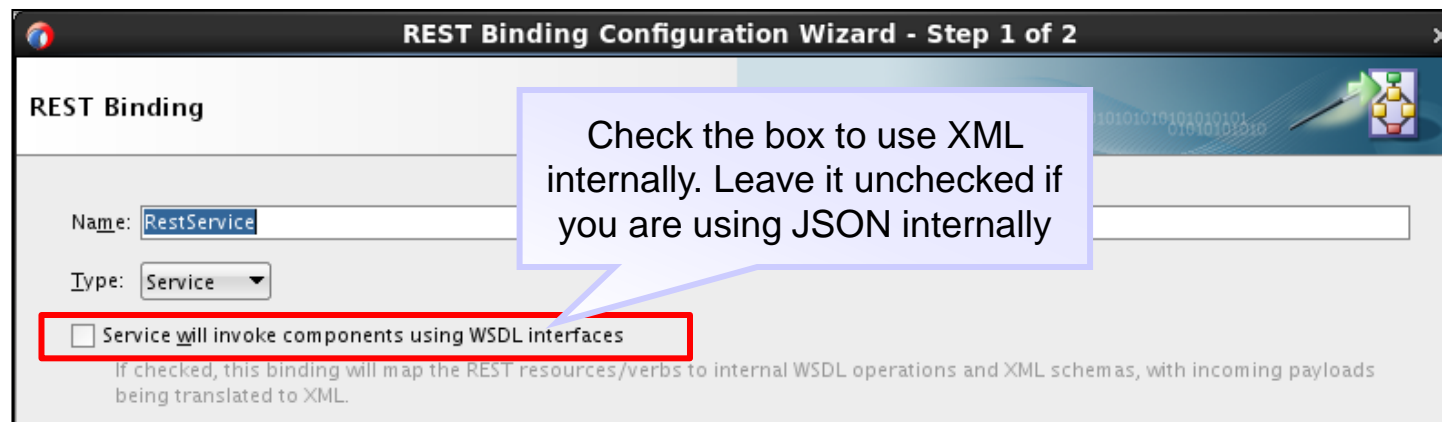


Follow the same steps as you would for creating any new Exposed Service interface.

Drag the REST Adapter to the Exposed Services swimlane.

Choose the Composite's Internal Data Format

- Choose if your service will use XML or JSON internally.
- While using JSON internally within the composite is possible, it is rarely required and the developer must be familiar with JSON and JavaScript to manipulate the variables in the BPEL component.
- It is easier and simpler to use XML internally.
- You can still consume and produce JSON if needed.
- If you will use XML internally, you will have to provide or generate an XSD to define the XML.



Configure the Binding

- You can configure the Resource Path(s), Operations and Parameters manually or use the Configuration Shortcut.

REST Binding Configuration Wizard - Step 2 of 2

Resources

Configuration Shortcut: +

Base URI:

Resources:

Resource Path
/CreditCheckRequest

Operation Bindings:

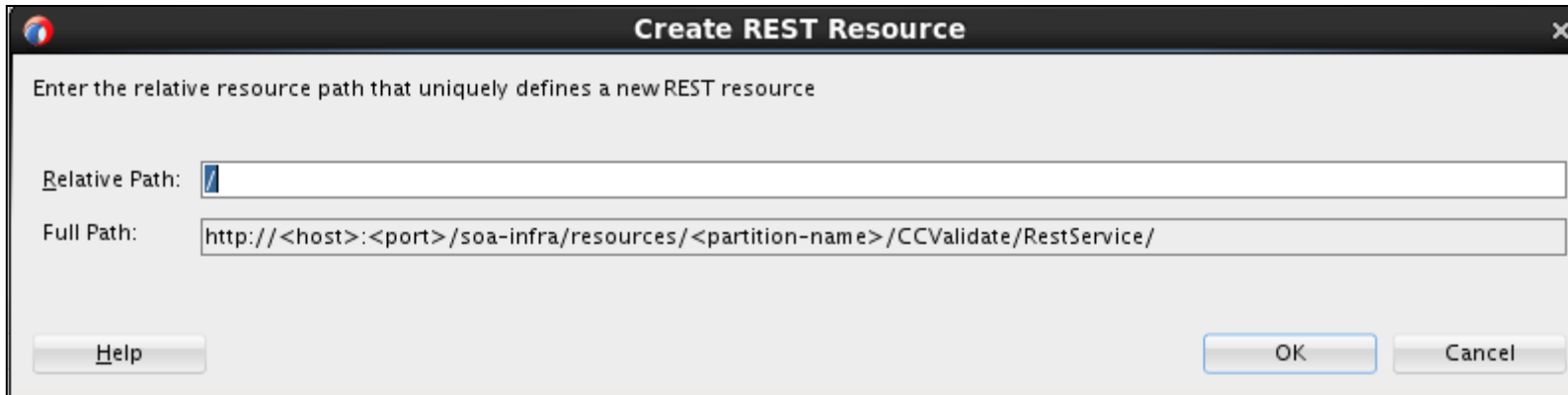
Operation	Resource Path	HTTP Verb	Complete
validateCC	/CreditCheckRequest	POST	yes
getOp	/CreditCheckRequest	GET	yes

If you have a WADL for the REST service, it can be used to pre-configure the binding.

Configuration Shortcut: +

Description:

Define the Resource Path(s)



A screenshot of a 'Create REST Resource' dialog box. The dialog has a title bar with a red, white, and blue icon on the left and a close button (X) on the right. The main area contains the instruction 'Enter the relative resource path that uniquely defines a new REST resource'. Below this, there are two text input fields. The first is labeled 'Relative Path:' and contains a single forward slash '/'. The second is labeled 'Full Path:' and contains the text 'http://<host>:<port>/soa-infra/resources/<partition-name>/CCValidate/RestService/'. At the bottom of the dialog, there are three buttons: 'Help' on the left, and 'OK' and 'Cancel' on the right.

Create REST Resource

Enter the relative resource path that uniquely defines a new REST resource

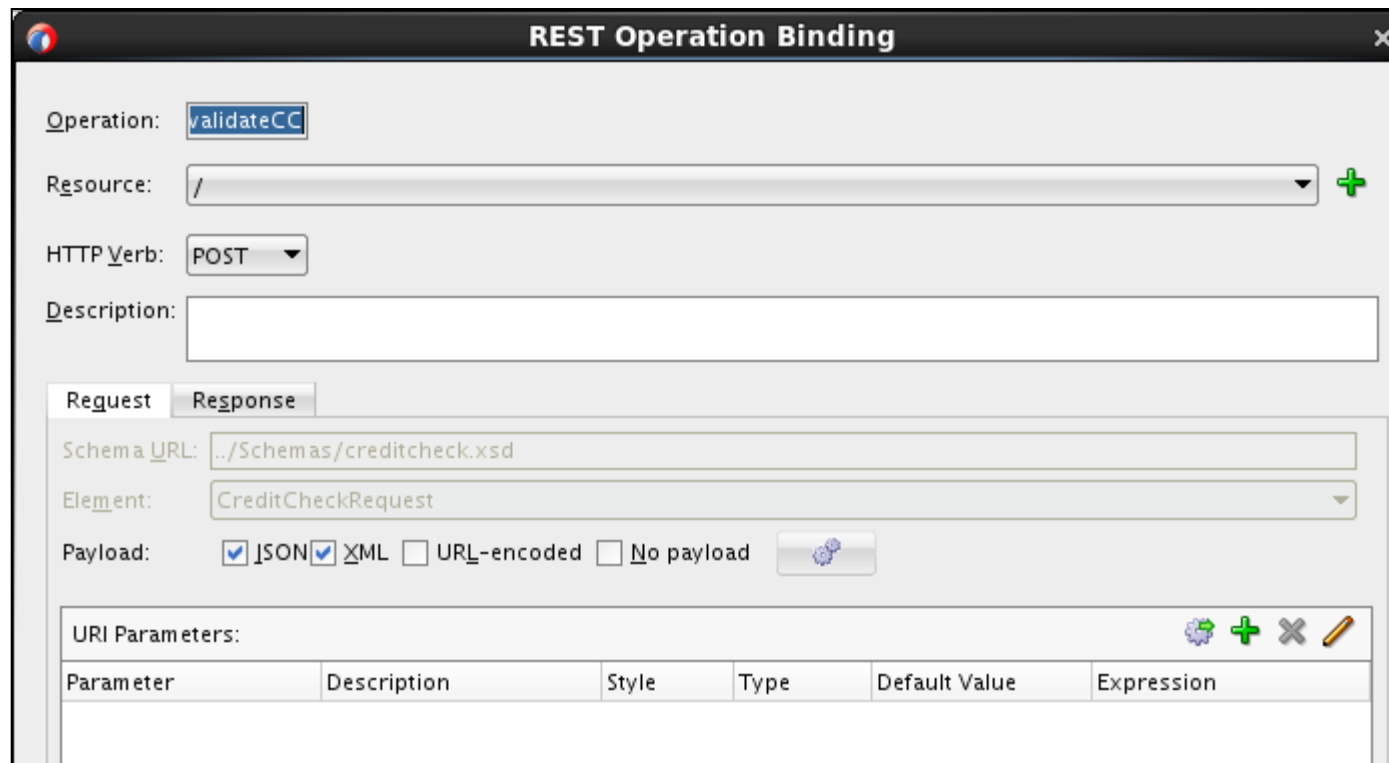
Relative Path: /

Full Path: http://<host>:<port>/soa-infra/resources/<partition-name>/CCValidate/RestService/

Help OK Cancel

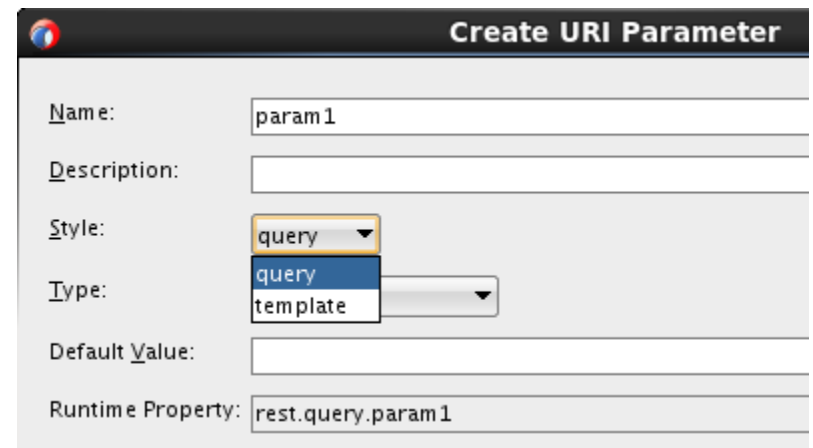
Define the Operation

- For each Resource Path operation, you must define the HTTP Verb, Request and Response Media types and any required parameters.



The **REST Operation Binding** dialog box is used to configure an API operation. It includes fields for the operation name, resource path, HTTP verb, and description. Below these are tabs for **Request** and **Response**. The **Request** tab is active, showing the schema URL, element name, and payload format (JSON, XML, URL-encoded, or No payload). At the bottom, there is a table for **URI Parameters**.

Parameter	Description	Style	Type	Default Value	Expression
-----------	-------------	-------	------	---------------	------------



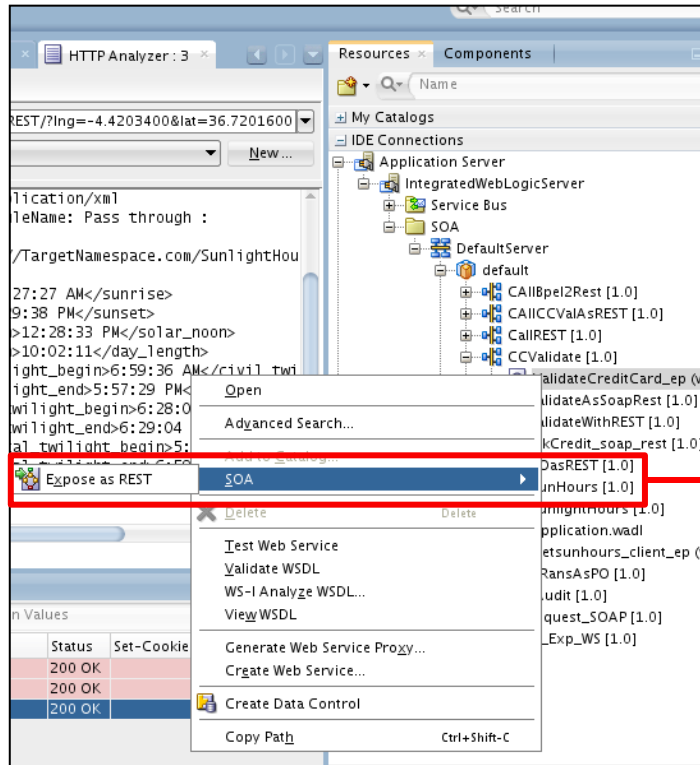
The **Create URI Parameter** dialog box is used to define a new URI parameter. It includes fields for the parameter name, description, style, type, default value, and runtime property.

Name:	param1
Description:	
Style:	query
Type:	query template
Default Value:	
Runtime Property:	rest.query.param1

Scenario 3: Create a REST External Reference for an External Service

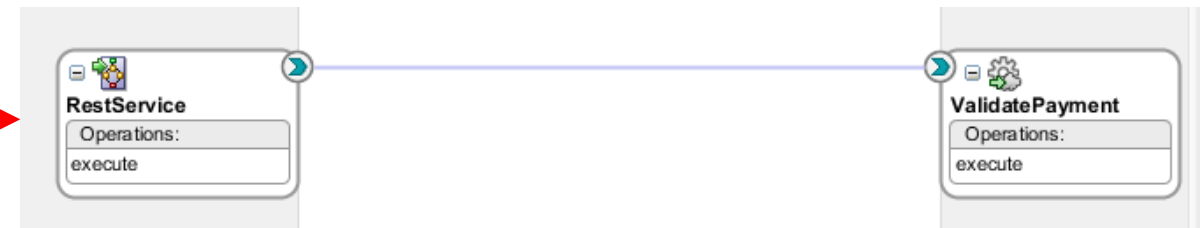
- You can create REST External References that call REST services by adding a REST adapter to the External References swimlane.
- These services could be other SOA composites or REST services external to SOA Suite.
- These services could be within your enterprise or outside on the internet.
- Configuring a REST External Reference requires the same choices and process as configuring a REST Exposed Interface:
 - Do you want the data in JSON or XML format inside the composite?
 - Do you have a WADL that describes the service or will you configure the REST binding manually?

Configuring the REST Adapter Through Shortcuts

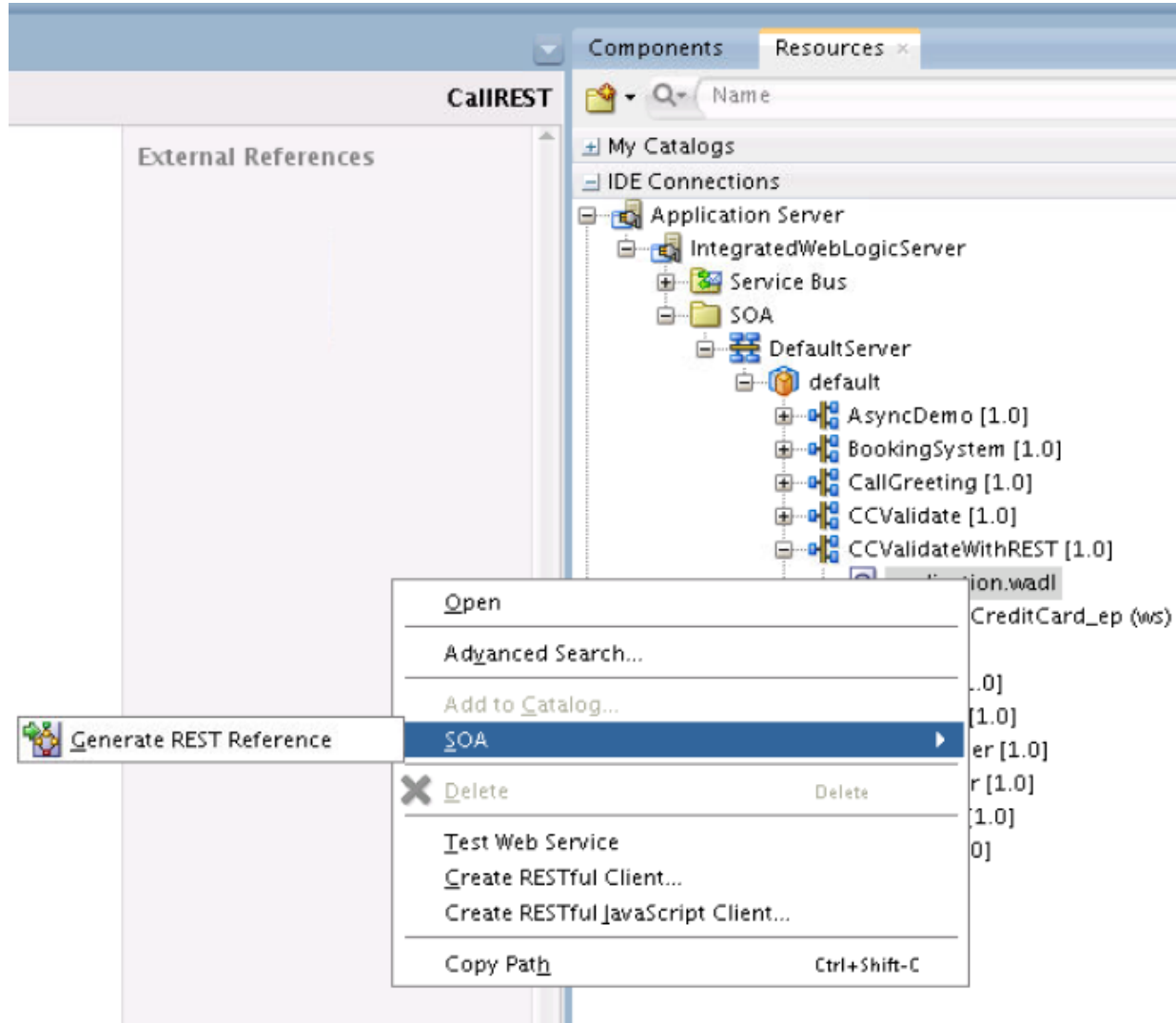


When exposing a SOAP service, a new REST interface service is created and passes the data to the SOAP service.

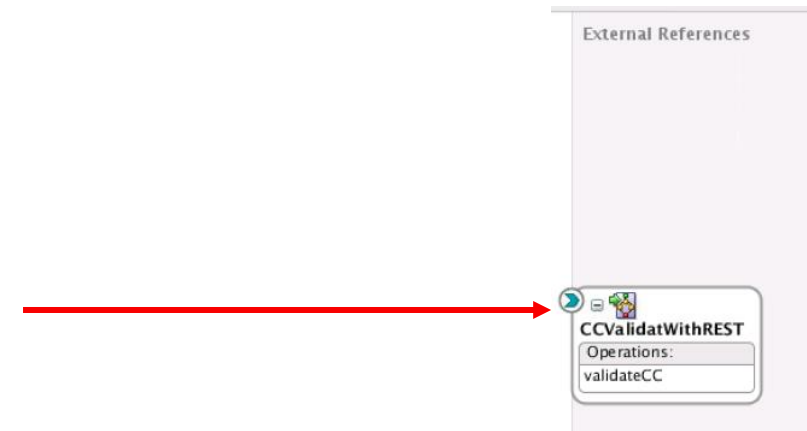
You can still customize the Resource Path, and choose the media type(s) the service provides.



Exposing an Existing REST Service as an External Reference



When exposing a REST service, the WADL is introspected, the REST service is automatically configured and the new REST External Reference is now available to be invoked in the composite.

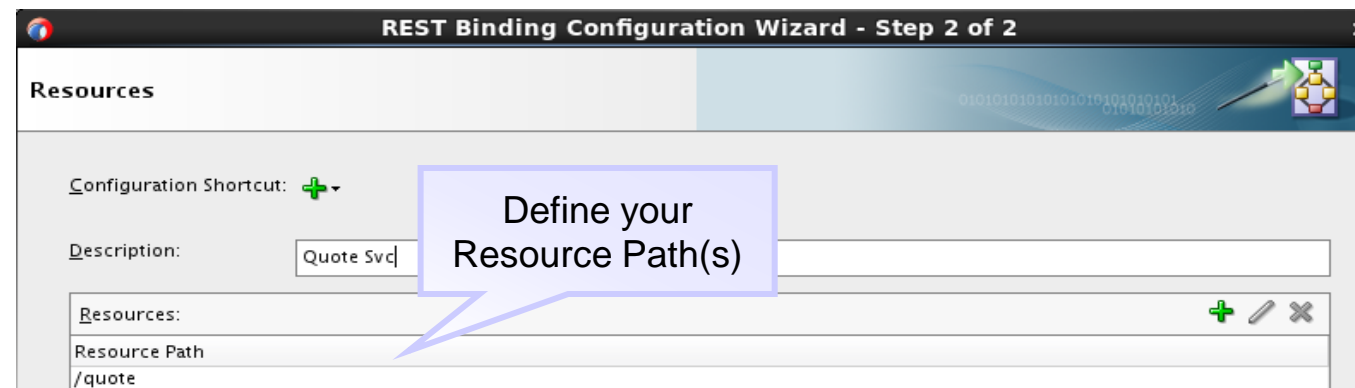
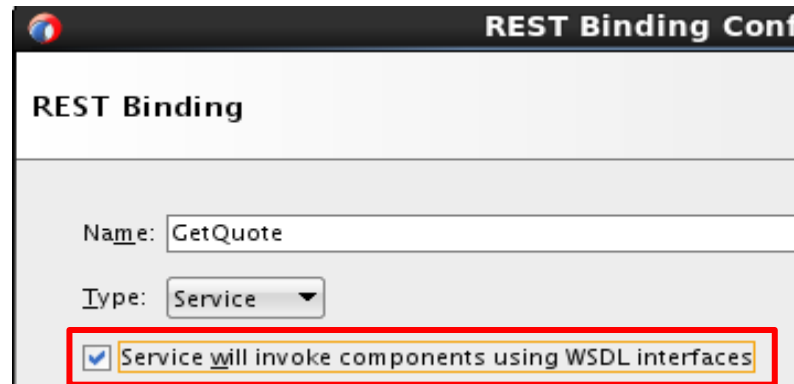


Configuring the REST Binding for WSDL and XML

- The easiest way to work with data consumed and produced by a REST interface is in XML.
- But, to do so, you need to have XSDs that describe the request and response data.
- If you do not have the XSDs, you can create them manually, which is time consuming and error prone.
- SOA Suite includes the Native Format Builder which will read non-XML data and produce XSDs.
- These XSDs can be used by the REST interfaces, as well as the other components, like the mediator and BPEL component.
- The XSDs could also be used to generate WSDL for a SOAP interface if you need to add that to your composite.

Using the Native Format Builder to Generate XSDs

If you choose to use XML for your REST Interface internally, select the check box:



Invoke the Native Format Builder

- You must supply an XSD for all request parameters and Response payloads.
- You can search for one on a file system or generate one using the Native Format Builder.

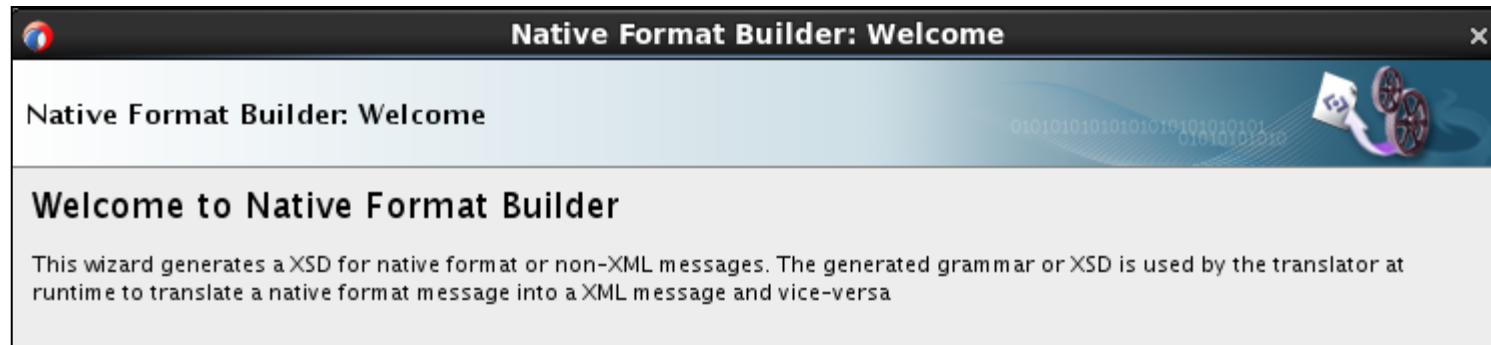


The screenshot shows a web interface for the Native Format Builder. It has two tabs: 'Request' and 'Response', with 'Response' currently selected. Below the tabs are two input fields: 'Schema URL' with the placeholder text '<<Not Configured>>' and 'Element' with a dropdown arrow. To the right of these fields are two icons: a magnifying glass (Browse) and a gear (Define Schema). Two callout boxes provide instructions: one points to the magnifying glass icon, and the other points to the gear icon.

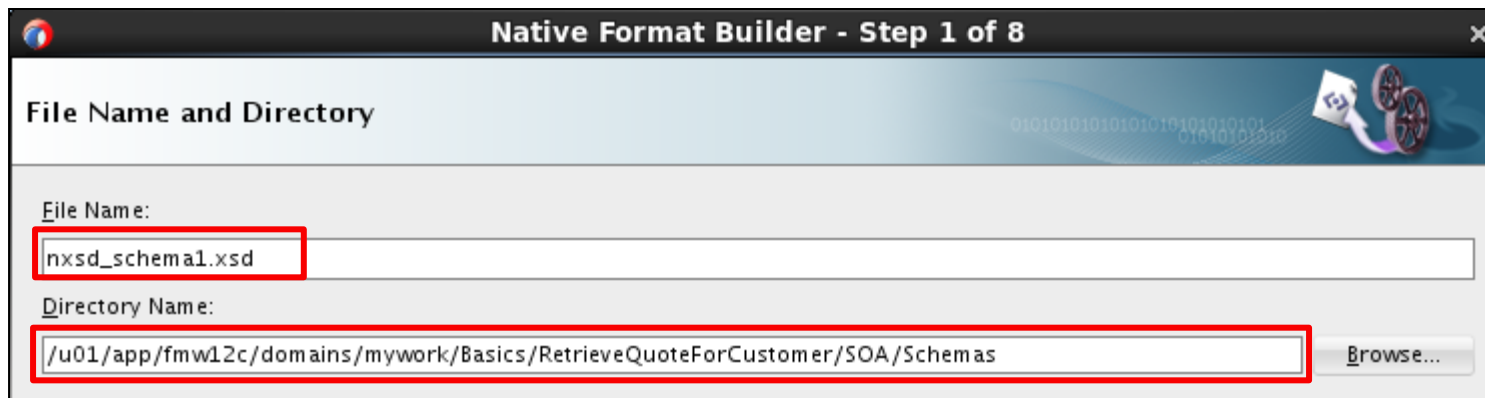
If you already have an XSD, click the Browse button to locate it.

Or Click the Define Schema for Native Format button to invoke the Native Format Builder.

The Native Format Builder

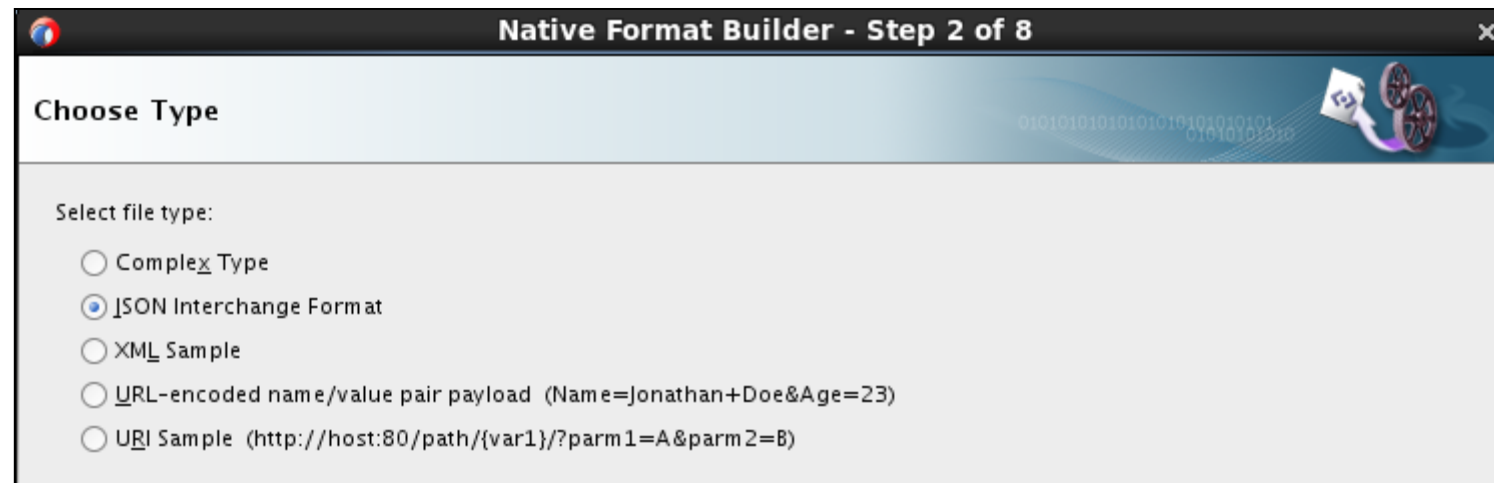


Define the name for your new schema and where it will be stored.



Choose the Native Data Type

Choose the native date type from which to generate the XSD:



Native Format Builder - Step 2 of 8

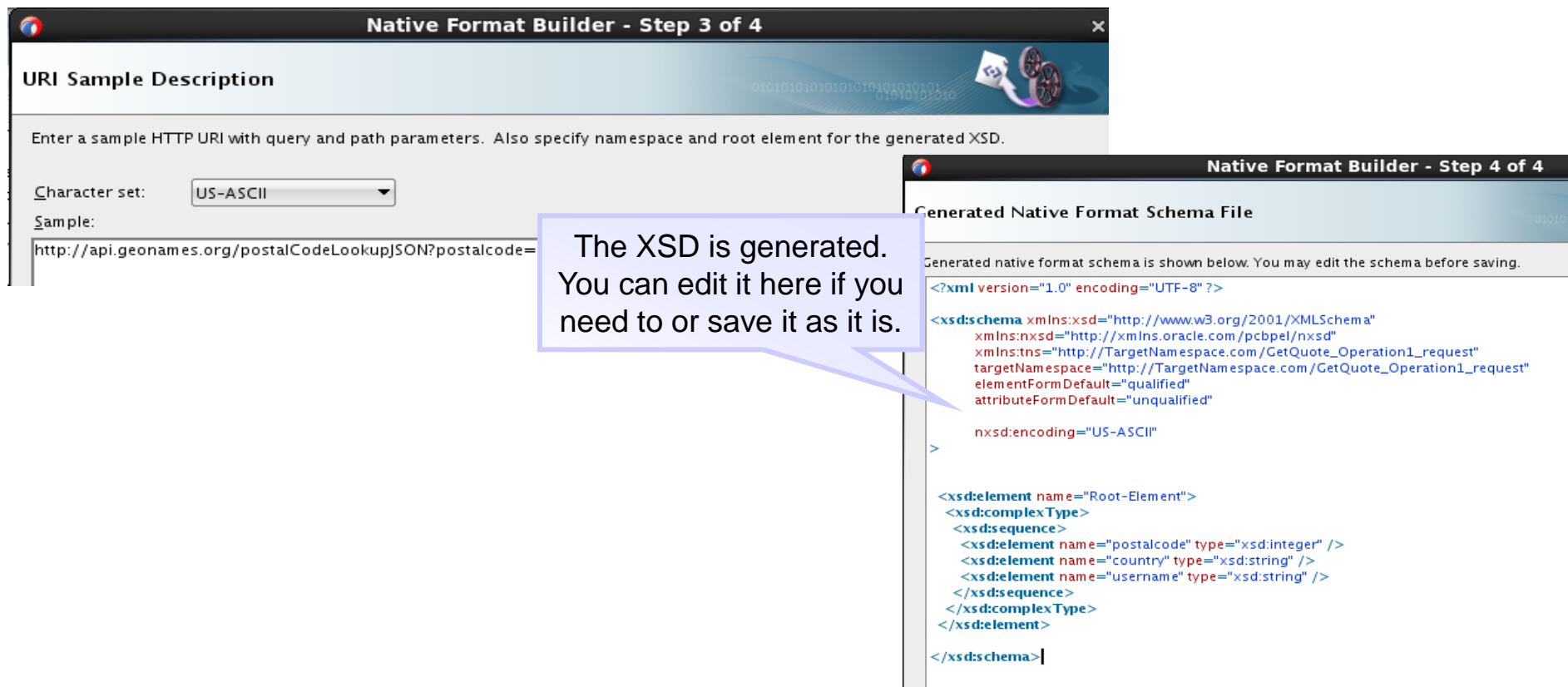
Choose Type

Select file type:

- ☐ Complex Type
- ☒ JSON Interchange Format
- ☐ XML Sample
- ☐ URL-encoded name/value pair payload (Name=Jonathan+Doe&Age=23)
- ☐ URI Sample (http://host:80/path/{var1}/?parm1=A&parm2=8)

URI Sample

If you have a URI for a REST service, enter it, choose a root-element Name and click next.



The image shows two overlapping windows from the 'Native Format Builder' tool. The background window is 'Step 3 of 4' titled 'URI Sample Description'. It contains a text area with the sample URI 'http://api.geonames.org/postalCodeLookupJSON?postalcode=' and a dropdown menu for 'Character set' set to 'US-ASCII'. The foreground window is 'Step 4 of 4' titled 'Generated Native Format Schema File'. It displays the generated XSD schema code. A purple callout bubble points to the XSD code with the text: 'The XSD is generated. You can edit it here if you need to or save it as it is.'

Native Format Builder - Step 3 of 4

URI Sample Description

Enter a sample HTTP URI with query and path parameters. Also specify namespace and root element for the generated XSD.

Character set:

Sample:

Native Format Builder - Step 4 of 4

Generated Native Format Schema File

Generated native format schema is shown below. You may edit the schema before saving.

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
  xmlns:tns="http://TargetNamespace.com/GetQuote_Operation1_request"
  targetNamespace="http://TargetNamespace.com/GetQuote_Operation1_request"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  nsxsd:encoding="US-ASCII"
>

  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="postalcode" type="xsd:integer" />
        <xsd:element name="country" type="xsd:string" />
        <xsd:element name="username" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

JSON Sample

- You can copy and paste the JSON into the Sample box, or if the JSON is in a file, you can browse and load that file.
- Choose a root-element name and click Next.
- The XSD is generated.

Native Format Builder - Step 3 of 4

JSON File Description

Enter a sample JSON payload or specify a sample JSON file location.

File name: Browse

Target namespace:

Root element:

Character set:

Sample:

```
{
  "lat": 47.5166667, "adminCode2": "708", "adminCode3": "70827", "adminName3": "Pinswang", "adminCode1": "07",
  "adminName2": "Politischer Bezirk Reutte", "lng": 10.6833333, "countryCode": "AT", "postalCode": "6600", "adminName1":
  "Tirol", "placeName": "Oberpinswang", "lat": 47.5333333, "adminCode2": "708", "adminCode3": "70826", "adminName3": "P
  flach", "adminCode1": "07", "adminName2": "Politischer Bezirk Reutte", "lng": 10.7166667, "countryCode": "AT", "postalCod
  e": "6600", "adminName1": "Tirol", "placeName": "Pflach", "lat": 47.5166667, "adminCode2": "708", "adminCode3": "70828",
  "adminName3": "Reutte", "adminCode1": "07", "adminName2": "Politischer Bezirk Reutte", "lng": 10.7166667, "countryCode"
  : "AT", "postalCode": "6600", "adminName1": "Tirol", "placeName": "Reutte", "lat": 47.4833333, "adminCode2": "708", "admin
  Code3": "70826", "adminName3": "Pflach", "adminCode1": "07", "adminName2": "Politischer Bezirk Reutte", "lng": 10.7, "coun
  tryCode": "AT", "postalCode": "6600", "adminName1": "Tirol", "placeName": "Unterletzen", "lat": 47.5166667, "adminCode2"
  : "708", "adminCode3": "70827", "adminName3": "Pinswang", "adminCode1": "07", "adminName2": "Politischer Bezirk Reutte
  ", "lng": 10.70065200805664, "countryCode": "AT", "postalCode": "6600", "adminName1": "Tirol", "placeName": "Unterpinsw
  ang", "lat": 47.500470170782684}}}
```

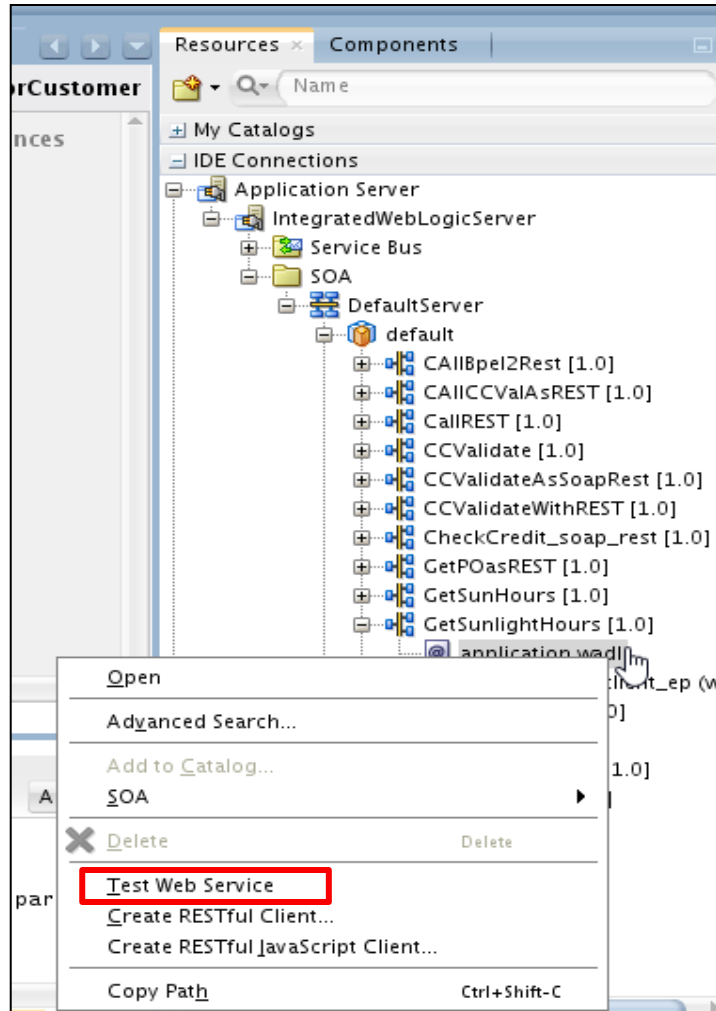
Native Format Builder - Step 4 of 4

Generated Native Format Schema File

Generated native format schema is shown below. You may edit the schema before saving.

```
<?xml version='1.0' encoding='UTF-8'?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://TargetNamespace.com/GetQuote_Operation1_
  <xsd:element name="Root-Element">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="postalCodes" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="adminCode2" type="xsd:integer"/>
              <xsd:element name="adminCode3" type="xsd:integer"/>
              <xsd:element name="adminName3" type="xsd:string"/>
              <xsd:element name="adminCode1" type="xsd:integer"/>
              <xsd:element name="adminName2" type="xsd:string"/>
              <xsd:element name="countryCode" type="xsd:string"/>
              <xsd:element name="postalCode" type="xsd:integer"/>
              <xsd:element name="adminName1" type="xsd:string"/>
              <xsd:element name="placeName" type="xsd:string"/>
              <xsd:element name="lat" type="xsd:double"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Testing the Component From JDeveloper



From the Resources Window,. Right-click the WADL for the REST service and select Test Web Service. This invokes the HTTP Analyzer in JDeveloper.

Examining the Results

The screenshot shows the HTTP Analyzer tool interface. The top toolbar includes icons for file operations and a status bar. The main window is titled "HTTP Analyzer : Unsent Message".

URL: `ple.com:7101/soa-infra/resources/default/GetSunlightHours!1.0/GetSunASREST/?lng=-4.4203400&lat=36.7201600`

WADL URI: `http://soa12c.example.com:7101/soa-infra...tHours!1.0/GetSunASREST/application.wadl` Select WADL ...

Operations: GET **Credentials:** <no credential> New ...

Request HTTP Headers + - X

Parameters

Parameter	Type	Value	Include
lat	decimal	36.7201600	<input checked="" type="checkbox"/>
lng	decimal	-4.4203400	<input checked="" type="checkbox"/>

body : No Content

Send Request Clear Request

Response HTTP Headers

Request is being edited

Content-Type: application/xml
X-HTTPAnalyzer-RuleName: Pass through :

```
<po xmlns="http://TargetNamespace.com/SunlightHou
<results>
  <sunrise>7:27:27 AM</sunrise>
  <sunset>5:29:38 PM</sunset>
  <solar_noon>12:28:33 PM</solar_noon>
  <day_length>10:02:11</day_length>
  <civil_twilight_begin>6:59:36 AM</civil_twili
  <civil_twilight_end>5:57:29 PM</civil_twili
  <nautical_twilight_begin>6:28:01 AM</nautic
  <nautical_twilight_end>6:29:04 PM</nautical
  <astronomical_twilight_begin>5:57:07 AM</astr
  <astronomical_twilight_end>6:59:58 PM</astr
</results>
<status/>
</po>
```

SOAP Structure HTTP Content REST Structure Hex Content Raw Message

Quiz



REST queries use _____ to describe resources.

- a. XML
- b. URIs
- c. SOAP
- d. JSON



Summary

In this lesson, you should have learned how to:

- Identify the format of REST queries
- List the differences between REST and SOAP
- Create and test a REST binding in a composite application



Practice 11 Overview

This practice covers the following topics:

- Exposing a REST interface
- Testing a REST interface
- Consuming a REST service

Practice 11-1 to 11-4: Overview

