TIBCO ActiveMatrix® Service Grid
REST Binding Development Guide

Software Release 3.3.1
July 2017

Document Update: August 2017
Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO and Two-Second Advantage are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

Copyright © 2010-2017 TIBCO Software Inc. All rights reserved.

TIBCO Software Inc. Confidential Information
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCO Documentation and Support Services</td>
<td>5</td>
</tr>
<tr>
<td>TIBCO ActiveMatrix Binding Type for REST Overview</td>
<td>7</td>
</tr>
<tr>
<td>REST Binding Type Key Terms</td>
<td>7</td>
</tr>
<tr>
<td>REST Binding Type Usage</td>
<td>8</td>
</tr>
<tr>
<td>REST Binding Type Key Features</td>
<td>8</td>
</tr>
<tr>
<td>Easy-to-Use Configuration GUI</td>
<td>8</td>
</tr>
<tr>
<td>Message Exchange Patterns</td>
<td>9</td>
</tr>
<tr>
<td>Error Handling</td>
<td>9</td>
</tr>
<tr>
<td>Complex XSD Constructs Mapping Rules</td>
<td>10</td>
</tr>
<tr>
<td>REST Binding Development</td>
<td>11</td>
</tr>
<tr>
<td>Payload Generation</td>
<td>11</td>
</tr>
<tr>
<td>Generating XML Payloads</td>
<td>11</td>
</tr>
<tr>
<td>Generating Badgerfish JSON Payloads</td>
<td>11</td>
</tr>
<tr>
<td>Overriding Media Types (For Service Only)</td>
<td>11</td>
</tr>
<tr>
<td>Configuring REST Bindings</td>
<td>12</td>
</tr>
<tr>
<td>Sending and Consuming HTTP Headers</td>
<td>14</td>
</tr>
<tr>
<td>Creating and Mapping Context Parameters</td>
<td>14</td>
</tr>
<tr>
<td>For REST Service Binding</td>
<td>15</td>
</tr>
<tr>
<td>Configuring for Request (Inbound) Flow</td>
<td>15</td>
</tr>
<tr>
<td>Configuring for Response (Outbound or Fault) Flow</td>
<td>16</td>
</tr>
<tr>
<td>For REST Reference Binding</td>
<td>16</td>
</tr>
<tr>
<td>Configuring for Request (Outbound) Flow</td>
<td>16</td>
</tr>
<tr>
<td>Configuring for Response (Inbound or Fault) Flow</td>
<td>17</td>
</tr>
<tr>
<td>Mapping HTTP Status Code and Status Message</td>
<td>18</td>
</tr>
<tr>
<td>Modifying a REST Resource Configuration File</td>
<td>18</td>
</tr>
<tr>
<td>Policies Supported</td>
<td>20</td>
</tr>
<tr>
<td>Sample Projects</td>
<td>21</td>
</tr>
<tr>
<td>Executing the Bookstore Sample</td>
<td>21</td>
</tr>
<tr>
<td>Importing the Bookstore Sample Project</td>
<td>22</td>
</tr>
<tr>
<td>Reviewing the WSDL that Defines the Service interface</td>
<td>22</td>
</tr>
<tr>
<td>Reviewing the Composite Configuration</td>
<td>23</td>
</tr>
<tr>
<td>Running the Bookstore Sample</td>
<td>23</td>
</tr>
<tr>
<td>Executing the MultipleComplexTypes Sample</td>
<td>24</td>
</tr>
<tr>
<td>Importing the MultipleComplexTypes Sample Project</td>
<td>24</td>
</tr>
<tr>
<td>Reviewing the Mediation Flow</td>
<td>24</td>
</tr>
<tr>
<td>Running the MultipleComplexTypes Sample</td>
<td>25</td>
</tr>
</tbody>
</table>
TIBCO Documentation and Support Services

Documentation for this and other TIBCO products is available on the TIBCO Documentation site. This site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, visit:

https://docs.tibco.com

Product-Specific Documentation

Documentation for TIBCO products is not bundled with the software. Instead, it is available on the TIBCO Documentation site.

The following documents form the documentation set:

- **Concepts**: Read this manual before reading any other manual in the documentation set. This manual describes terminology and concepts of the platform. The other manuals in the documentation set assume you are familiar with the information in this manual.
- **Development Tutorials**: Read this manual for a step-by-step introduction to the process of creating, packaging, and running composites in TIBCO Business Studio.
- **Composite Development**: Read this manual to learn how to develop and package composites.
- **Java Component Development**: Read this manual to learn how to configure and implement Java components.
- **Mediation Component Development**: Read this manual to learn how to configure and implement Mediation components.
- **Mediation API Reference**: Read this manual to learn how to develop custom Mediation tasks.
- **Spring Component Development**: Read this manual to learn how to configure and implement Spring components.
- **WebApp Component Development**: Read this manual to learn how to configure and implement Web Application components.
- **REST Binding Development**: Read this manual to learn how to configure and implement REST components.
- **Administration Tutorial**: Read this manual for a step-by-step introduction to the process of creating and starting the runtime version of the product, starting TIBCO ActiveMatrix servers, and deploying applications to the runtime.
- **Administration**: Read this manual to learn how to manage the runtime and deploy and manage applications.
- **Hawk ActiveMatrix Plug-in User’s Guide**: Read this manual to learn about the Hawk plug-in and its optional configurations.
- **Error Codes**: Read this manual to know more about the error messages and how you could use them to troubleshoot a problem.
- **Installation and Configuration**: Read this manual to learn how to install and configure the software.
- **Release Notes**: Read this manual for a list of new and changed features, steps for migrating from a previous release, and lists of known issues and closed issues for the release.

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:
If you already have a valid maintenance or support contract, visit this site:
https://support.tibco.com

Entry to this site requires a user name and password. If you do not have a user name, you can request one.

How to Join TIBCO Community

TIBCO Community is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCO Community offers forums, blogs, and access to a variety of resources including product wikis that provide in-depth information, white papers, and video tutorials. In addition, users can submit and vote on feature requests via the Ideas portal. For a free registration, go to https://community.tibco.com.
TIBCO ActiveMatrix Binding Type for REST Overview

TIBCO ActiveMatrix Binding Type for REST allows you to map SCA services to REST, so that thin clients like scripting, mobile, and Web clients can directly invoke these services.

TIBCO ActiveMatrix Binding Type for REST makes client development simpler and cheaper by not requiring a SOAP stack on client side. Clients can use HTTP methods such as GET, POST, PUT, and POST with XML or JSON to invoke backend SCA services. During configuration, users can choose the XML, JSON, or BJSON media type in TIBCO Business Studio UI.

When you configure a SOAP service, WSDL becomes the contract between service provider and consumer. In contrast, REST service providers do not use WSDL but use out of band mechanisms such as sample payloads to communicate with REST service consumers. This release of TIBCO ActiveMatrix Binding Type for REST does not include tools for generating these payloads but includes an example and documentation.

REST Binding Type Key Terms

HTTP Connector, Context Root, Media Type, Path Parameters and Query Parameters are few key terms used while discussing about TIBCO ActiveMatrix Binding Type for REST.

See the TIBCO ActiveMatrix Service Grid documentation for a discussion of general ActiveMatrix terms. The following list presents the key terms for TIBCO ActiveMatrix Binding Type for REST.

HTTP Connector
Name of the HTTP Connector resource instance that provides the HTTP transport for Binding Type for REST. Both HTTP and HTTPS are supported. Default is HTTP. You define and name the HTTP Connector at design time. At runtime, you need to create a resource of type HTTP Connector and assign it the name you used at design time.

Context Root
Defines the base path for the URLs exposed by the REST binding.

Media Type
Format of the payload that ActiveMatrix Binding Type for REST accepts and produces. On the reference side, XML and Standard JSON are supported. On the service side, XML, Standard JSON, and Badgerfish JSON are supported.

Path
You can specify Path as part of configuration. Path can be any URI on which a given operation can be exposed.

Path Parameters
Path parameters can be configured on the service side and reference side.

On the service side, path parameters should map to the part name defined in the WSDL. Path parameters can be defined on the operation by using the Path field in the UI. For example, if you want to invoke a backend service operation getBookByTitle(title), you can configure the path as /book/{title}. Path parameters are supported by parts that are simple types. On the reference side, path parameters can be configured by adding them in the 'Resource Path' field (for example, '/{<pathParameter1>}'). For the above getBookByTitle(title) operation, 'title' can be added as a path parameter using syntax: /book/{title}. Here, /book is the Resource Path and /{title} is the Path Parameter.

Query Parameters
Query parameters can be configured on the service side and reference side.

On the Service side, query parameters are not configured as part of the path for the operation. TIBCO ActiveMatrix Binding Type for REST expects that the query parameter name matches the part name of
the WSDL operation. If you want to use a query parameter, the part name must be a simple type such as string, boolean, int, and so on.

On the reference side, you can configure a query parameter by adding them in the "Request Parameters" section of the REST Resource Configuration file.

You cannot add Content-Type or Accept as a Request Parameter.

**REST Binding Type Usage**

TIBCO ActiveMatrix Binding Type for REST allows you to integrate your SCA services with clients that use HTTP instead of SOAP to invoke services.

ActiveMatrix service development typically starts with WSDL, which defines the service interfaces. Developers expose SOAP or JMS services by adding SOAP or JMS bindings on a promoted component service.

TIBCO ActiveMatrix Binding Type for REST allows you to expose those services as REST services that can consume Badgerfish JSON, Standard JSON, or XML. You can add multiple bindings and multiple types of bindings on the same composite services. That means the same service can expose SOAP, JMS, and REST interfaces to service consumers at the same time.

Typical use cases include the following.

- Mobile clients have to consume an ActiveMatrix SCA service.
- Web clients or scripting clients (thin clients) participate in SCA.
- Mashups or web sites have to expose services as APIs.

The REST bindings are especially helpful in the following situations.

- Mobile devices need to interact with back-end applications and services.
- Mobile application developers find it difficult to program the SOAP stack on the client side.
- Developers use modern scripting languages like JavaScript and Ruby, which provide first class support for JSON and XML processing.

In these cases, TIBCO ActiveMatrix Binding Type for REST allows the clients to easily invoke ActiveMatrix SCA services.

If you want to expose already existing services, you might have to use mediation. When an existing ActiveMatrix service is using WSDLs with multiple parts of complex type, the mediation service can normalize the WSDL to have a single part of complex type. See Executing the MultipleComplexTypes Sample on page 24.

**REST Binding Type Key Features**

Using a configuration GUI and a robust error handler, TIBCO ActiveMatrix Binding Type for REST allows users to map SCA services as REST services and also allows users to consume a REST service using a REST reference.

**Easy-to-Use Configuration GUI**

TIBCO ActiveMatrix Binding Type for REST provides a custom binding palette to add, configure, and remove a REST binding using the TIBCO Business Studio.

This easy-to-use interface simplifies configuration. You can add, edit, or remove bindings. In addition, you can perform the following tasks specific to a service or a reference:
For a Service

- Specify a name, context root, HTTP connector and media type for the binding.
- Configure the operation to use one of the supported HTTP methods.
  - Use HTTP GET or HTTP DELETE when the target WSDL operation has parts (single or multiple parts) of simple type.
  - Use HTTP POST or HTTP PUT to send XML, Standard JSON, and Badgerfish JSON payloads. Only one complexType is allowed when the input message is a multi-part message. Use a mediation component if your source has multiple complex types.
- Use a mediation component if you have to map a WSDL that uses multiple parts of complex type. See Complex XSD Constructs Mapping Rules on page 10.
- You can choose Standard JSON, Badgerfish JSON, or XML as the media type.
  - **Standard JSON** TIBCO ActiveMatrix Binding Type for REST can consume and produce JSON as defined by the standard convention.
  - **Badgerfish JSON** TIBCO ActiveMatrix Binding Type for REST can consume and produce JSON as defined by the Badgerfish convention. Badgerfish JSON is being used because it maps XML constructs such as namespaces to JSON.
  - **XML** TIBCO ActiveMatrix Binding Type for REST can consume and produce XML. The XML payload must be schema compliant.

For a Reference

Specify a name, description, REST resource configuration file, HTTP Client, enable or disable Pass Thorugh Mode. For more information on modifying REST resource configuration file, refer to Modifying a REST Resource Configuration File on page 18.

Message Exchange Patterns

Clients of TIBCO ActiveMatrix Binding Type for REST can invoke backend services that are exposing IN-ONLY or IN-OUT message exchange patterns using HTTP operations (GET, PUT, POST, DELETE).

For a Service

- IN-ONLY operations return HTTP code 200 OK for success or an HTTP error code in case of failure.
- IN-OUT operations return a response or a fault in the HTTP body for success or failure. An HTTP error code is returned for protocol errors.

For a Reference

Configuring Out-Only operation in REST Resource Configuration file results in addition of dummy queryParameter Named ‘AmxInOutBoolean’ for that operation.

Error Handling

TIBCO ActiveMatrix Binding Type for REST returns error in response body in case there is fault in request or a component.

TIBCO ActiveMatrix Binding Type for REST handles errors as follows:

**Service**

- System errors such as invalid requests are returned as protocol errors, that is, HTTP error codes.
- When a wired service returns a SOAP fault, the REST binding returns a 200 OK code by default. A fault message is returned as a response body.
The component that implements the WSDL service can override the HTTP response code by using a context parameter named `HTTP_RESP_CODE`. This parameter is of type `int`.

If a component throws an undeclared-fault or a runtime exception, TIBCO ActiveMatrix Binding Type for REST returns an Internal Server Error with HTTP Code 500 and an `HTTP_RESP_CODE`. If any context variables are set, they are ignored.

Reference

- All errors for a specific operation can be configured in the REST resource configuration file.
- Implementation Type consuming REST reference can receive HTTP Status Code and Status Message of the response received using Context Parameters.
  
  Direction: Output
  
  Data Type: `int` (For `statusCode`) and `String` (for `Status Message`)
  
  Header Name: `statusCode` (For HTTP Status Code) / `statusMsg` (For HTTP Status Message)
  
  Context Parameter Name: `statusCode` (For HTTP Status Code) / `statusMsg` (For HTTP Status Message)

**Complex XSD Constructs Mapping Rules**

If backend services are using complex XSD constructs for WSDL operation signatures, such as multiple parts with complex types, you can use a mediation component between the Implementation Type for REST component and the backend component.

Follow these rules when mapping WSDL operation arguments (message parts) to HTTP operations.

- Multi-part Operation
  
  - Simple types, or built-in simple XSD types such as `string`, `float`, `boolean`, `integer` and so on, must be passed as query or path parameters.
  
  - The name of the query parameter or path parameter must match the part name
  
  - Complex types, such as `xsd:complexType` must be passed as HTTP body

  TIBCO ActiveMatrix Binding Type for REST supports backend WSDL operations with only one part of `complexType` in the operation signature for multipart WSDL Executing the [MultipleComplexTypes Sample](#) on page 24 illustrates how to use mediation to expose a multi-part WSDL to REST clients.

- Single-part Operation
  
  - Simple types, or built-in simple XSD types such as `string`, `float`, `boolean`, `integer` and so on, must be passed as query or path parameters.
  
  - Complex types, such as `xsd:complexType` must be passed as HTTP body.
REST Binding Development

Create and configure a binding for REST with TIBCO Business Studio and package the binding into a distributed application archive (DAA).

TIBCO ActiveMatrix Business Studio is a standards-based, unified business process modeling and development environment for modeling, developing, and deploying business process applications. TIBCO ActiveMatrix Binding Type for REST is integrated with ActiveMatrix Business Studio so you can configure and test the REST bindings from there.

For more information about TIBCO Business Studio, see the Workbench User Guide in the Workbench online help. To view the online help, select Help > Help Contents.

After you configure and test the REST binding, you can create and deploy a distributed application archive from your project. See the TIBCO ActiveMatrix documentation for details.

Payload Generation

TIBCO ActiveMatrix Binding Type for REST supports XML, Standard JSON, and Badgerfish JSON payloads. You can generate an XML file from a WSDL file, and then using a tool included with TIBCO ActiveMatrix Binding Type for REST, you can generate a Badgerfish JSON payload.

Generating XML Payloads

Generate XML file based on where the part types definition are defined.

- Part types are defined in the XSD, imported from WSDL
  Right-click the .xsd file and select Generate > XML file.
- Part types are defined in WSDL
  In Eclipse, make a copy of the WSDL and change the extension to .xsd. Or, right-click the .xsd file and select Generate > XML file.

The generated XML file is a valid XML payload for methods exposed over HTTP POST.

Generating Badgerfish JSON Payloads

You can generate the payload for JSON with xmltojson tool, included in the installation.

Procedure

1. Go to TIBCO_HOME/amx/<version_number>/samples/rest/tools.
2. Run xmltojson tool. You pass in the XML file you generated from Eclipse, see Generating XML Payloads on page 11.

Result

For more information, see the readme of the tool.

Overriding Media Types (For Service Only)

You can override the media types to be consumed or produced by using HTTP headers.

The Media Type configured in the UI serves as the default media type for both the HTTP Request body content type and for the HTTP Response body content type.
You need to specify this information only if your REST client has to override the media type on a per message basis. In most cases, the configuration specifies the media type that is used throughout the application.

Override the content type based on the message type.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Set the Content-Type header in the HTTP RequestClients to override the request body content type. The values are:</td>
</tr>
<tr>
<td></td>
<td>- Content-Type: application/json</td>
</tr>
<tr>
<td></td>
<td>- Content-Type: application/xml</td>
</tr>
<tr>
<td></td>
<td>- Content-Type: application/bjson</td>
</tr>
</tbody>
</table>

| Response | Set the Accept header in the HTTP RequestClients to override the response content type. The values are: |
|          | - Accept: application/json |
|          | - Accept: application/xml |
|          | - Accept: application/bjson |

### Configuring REST Bindings

Specify various details after adding the REST binding type to the composite service or reference using TIBCO Business Studio.

#### Prerequisites

Install all four profiles of ActiveMatrix Service Grid or ActiveMatrix Service Bus.

You add a REST binding to the composite service or reference from the TIBCO Business Studio Composite Editor.

#### Procedure

1. Import an existing project or create a new project in TIBCO ActiveMatrix Business Studio. See the TIBCO ActiveMatrix documentation set.
2. Add a REST binding from the canvas view or the properties view.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canvas view</td>
<td>Right-click the service or reference and select Add &gt; REST Binding from the popup menu.</td>
</tr>
</tbody>
</table>
| Properties view | - Click the service or reference on the canvas.  
|               | - In the Properties view, click the Bindings tab.  
|               | - Click the Add Binding... button. |
3. Select the service or reference and display its properties.
4. For a promoted service, specify the following details in the right pane of the Properties View.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the REST binding.</td>
</tr>
<tr>
<td>HTTP Connector</td>
<td>HTTP Connector and Context Root together define the URL that is used at runtime. You define and name the HTTP Connector at design time. At runtime, you need to create a resource of type HTTP Connector and assign it the name you used at design time.</td>
</tr>
<tr>
<td>Context Root</td>
<td>HTTP Connector and Context Root together define the URL that is used at runtime.</td>
</tr>
<tr>
<td>Media Type</td>
<td>Select Standard JSON, Badgerfish JSON, or XML from the pull-down menu to specify the media type for the request or the response message.</td>
</tr>
<tr>
<td>Exclude namespaces from response</td>
<td>Excludes namespaces from the response message. This option is displayed only when Media Type is set to Badgerfish JSON.</td>
</tr>
</tbody>
</table>

5. For a promoted reference, specify the following details in the right pane of the Properties View.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the REST binding.</td>
</tr>
<tr>
<td>Description</td>
<td>Short description of the REST binding.</td>
</tr>
<tr>
<td>Media Type</td>
<td>(read-only field). The media type can be set from the REST resource configuration file.</td>
</tr>
<tr>
<td>Rest resource configuration file</td>
<td>Location of the REST resource configuration file. A resource configuration file is not set by default. To create a resource configuration file, click the -not set- link or click the picker icon and then click Create New. When you click on Finish in this wizard, a new resource configuration file (.rrc) is created with default media type (JSON) and resource base URI. The .rrc file must be placed in the Service Descriptor folder of the SOA project.</td>
</tr>
<tr>
<td>HTTP Client</td>
<td>Select the HTTP client.</td>
</tr>
<tr>
<td>Enable pass through mode</td>
<td>Enables the pass through mode. In the pass through mode, a fixed WSDL is configured on the REST binding. In the pass through mode, ActiveMatrix Binding Type for REST behaves like the HTTP Binding Type.</td>
</tr>
</tbody>
</table>
Sending and Consuming HTTP Headers

You can send and consume HTTP headers from the REST operation invocations, using context parameters on REST reference and service bindings. The values populated by the REST binding, map HTTP Transport headers to context parameters.

In the case of REST reference bindings, using context mapping, the values set by the Implementation type can be sent as HTTP Headers as part of the REST request. Also, the HTTP Headers received as part of the REST response, can be made available to the Implementation Type.

In the case of REST service bindings, using context mapping, the values of the HTTP Transport Headers can be made available to the Implementation Type (for example, Java IT). Also, the values set by the Implementation Type can be sent as HTTP Transport Headers as part of the REST response.

The Content-Type or Accept headers are not supported in context parameters and the REST Resource Configuration (RRC) file. That is, you cannot add a context parameter with the name as Content-Type or Accept. In the RRC file, you cannot add Content-Type or Accept as a Request Parameter.

For details on configuring the Inbound, Outbound, and Fault messages for service and reference, refer to the following sections. A sample project is provided as part of the installation in TIBCO_HOME/amx/<version>/samples/rest/samples/rest.context to elaborate on the same. Refer to the /rest.context/Readme.pdf for details.

Creating and Mapping Context Parameters

You can add context parameters to REST bindings in the Context Parameters section of the General tab.

Procedure

1. Navigate to the General tab > Context Parameters section of a Promoted Service or Reference.
2. Add a new Context Parameter; select the Operation(s) it applies to, the Direction (Input, Output or Fault), and the Type (Basic or Bag). For Basic Context Parameters, select a Definition, to describe the data type of the parameter.

For instance, bookNameCP is a Basic string context parameter that applies to the input flow of the getBookList operation. The name bookNameCP is used for context parameter mapping in the next step, and is also referred to by IT Implementation.

3. Navigate to the Bindings tab, in the REST Binding Context Parameter Mapping section and specify the HTTP header name in the Header Name column.
4. If the selected type is Basic, specify the HTTP header name in the **Header Name** column. The specified Header Name appears as a HTTP Header on the wire and the context Parameter name is used by Java IT for the Get and Set methods. For instance, the Basic string context parameter bookNameCP is mapped to HTTP Header Name bookName. That is, when REST service binding intercepts a request, it retrieves the value of the HTTP Header bookName and makes it available to IT Implementation as a value of context parameter bookNameCP.

5. Generate Java IT Implementation and add the following declaration to the Implementation class:

```java
//org.osoa.sca.annotations.Context
@Context
//com.tibco.amf.platform.runtime.extension.context.ComponentContext
public ComponentContext componentContext;
```

**For REST Service Binding**

To configure the inbound and outbound messages, see:

- *Configuring for Request (Inbound) Flow* on page 15
- *Configuring for Response (Outbound or Fault) Flow* on page 16

**Configuring for Request (Inbound) Flow**

You can configure inbound messages received on a service by configuring the following:

- Supported Type: Basic, Bag
- Supported Header Source: HTTP_HEADERS
- Direction: Inbound

Add the following code after receiving the service operation response:

To retrieve a 'Basic' context parameter:

```java
//Retrieve requestContext from componentContext which contains the inbound context parameters from Service-side
RequestContext originalRequestContext = (RequestContext) componentContext.getRequestContext();
originalRequestContext.getParameter("bookNameCP", String.class));
```

The return value of above `getParameter()` call corresponds to the HTTP Header value of the bookName header (corresponding to context parameter bookNameCP) in the incoming REST Request.

To retrieve context parameter 'Bag':

```java
//Retrieve requestContext from componentContext which contains the inbound context parameters from Service-side
```
RequestContext originalRequestContext = (RequestContext) componentContext.getRequestContext();

HashMap<String, String> requestHeadersAllCP_service_map = (HashMap<String, String>) originalRequestContext.getParameter("requestHeadersAllCP", Map.class);

The HashMap requestHeadersAllCP_service_map contains all the HTTP Headers (user-defined and native HTTP Headers, corresponding to context parameter requestHeadersAllCP) in the incoming REST Request.

Configuring for Response (Outbound or Fault) Flow

You can configure outbound messages sent from a service by configuring the following:

- Supported Type: Basic, Bag
- Supported Header Source: HTTP_HEADERS
- Direction: Outbound / Fault

Add the following code.

To set a 'Basic' context parameter:

```java
//Create a Mutable callbackContext (Response flow) from the original RequestContext
MutableCallbackContext originalCallbackContext = (MutableCallbackContext) originalRequestContext.createCallbackContext();

originalCallbackContext.setParameter("bookCategoryCP", String.class, "Classic");
```

The value Classic is set for the HTTP Header bookCategory (corresponding to the context parameter bookCategoryCP) in the outgoing REST Response.

To set a context parameter 'Bag':

```java
//Create a Mutable callbackContext (Response flow) from the original RequestContext
MutableCallbackContext originalCallbackContext = (MutableCallbackContext) originalRequestContext.createCallbackContext();

responseHeadersAllCP_service_map = new HashMap<String, String>();
responseHeadersAllCP_service_map.put("bookAuthor", "Harper Lee");
responseHeadersAllCP_service_map.put("bookPublishYear", "1960");
originalCallbackContext.setParameter("responseHeadersAllCP", Map.class, responseHeadersAllCP_service_map);
```

The contents of the HashMap responseHeadersAllCP_service_map will be set as HTTP Headers (user-defined, corresponding to context parameter responseHeadersAllCP) in the outgoing REST Response. Setting native HTTP Headers is not permitted.

For REST Reference Binding

To configure the inbound and outbound messages, see the following sections.

- Configuring for Request (Outbound) Flow on page 16
- Configuring for Response (Inbound or Fault) Flow on page 17

Configuring for Request (Outbound) Flow

You can configure outbound messages sent from a reference by configuring the following:

- Supported Type: Basic, Bag
- Supported Header Source: HTTP_HEADERS
- Direction: Outbound

Add the following code.
To set a 'Basic' context parameter:

```java
//Create a new Mutable requestContext from componentContext to set context parameters for Reference-side
MutableRequestContext createMutableRequestContext = componentContext.createMutableRequestContext();
createMutableRequestContext.setParameter("bookNameCP", String.class, "How to Kill a Mockingbird");
```

The value How to Kill a Mockingbird is set for the HTTP Header bookName (corresponding to context parameter bookNameCP) in the outgoing REST Request.

To set a context parameter 'Bag':

```java
//Create a new Mutable requestContext from componentContext to set context parameters for Reference-side
MutableRequestContext createMutableRequestContext = componentContext.createMutableRequestContext();
Map<String, String> requestHeadersAllCP_reference_map = new HashMap<String, String>(){
    requestHeadersAllCP_reference_map.put("bookAuthor", "Harper Lee");
    requestHeadersAllCP_reference_map.put("bookPublishYear", "1960");
    createMutableRequestContext.setParameter("requestHeadersAllCP", Map.class, requestHeadersAllCP_reference_map);
```

The contents of the HashMap requestHeadersAllCP_reference_map are set as HTTP Headers (user-defined, corresponding to context parameter requestHeadersAllCP) in the outgoing REST Request.

### Configuring for Response (Inbound or Fault) Flow

You can configure inbound messages received on a reference ('out|fault' part of "in-out" MEP) by configuring the following:

- Supported Type: Basic, Bag
- Supported Header Source: HTTP_HEADERS
- Direction: Inbound/Fault

Add the following code after receiving the service operation response:

To retrieve a 'Basic' context parameter:

```java
//Retrieve the callbackContext (Response/Fault flow) from the mutableRequestContext
CallbackContext callbackContext = createMutableRequestContext.getCallbackContext();
callbackContext.getParameter("bookCategoryCP", String.class));
```

The return value of above getParameter() call corresponds to the HTTP Header value of the bookCategory header (corresponding to the context parameter bookCategoryCP) in the incoming REST Response.

To retrieve a context parameter 'Bag':

```java
//Retrieve the callbackContext (Response/Fault flow) from the mutableRequestContext
CallbackContext callbackContext = createMutableRequestContext.getCallbackContext();
HashMap<String, String> responseHeadersAllCP_reference_map = (HashMap<String, String>)callbackContext.getParameter("responseHeadersAllCP", Map.class);
```

The HashMap responseHeadersAllCP_reference_map contains all HTTP Headers (user-defined and native HTTP Headers, corresponding to context parameter responseHeadersAllCP) in the incoming REST Response.
Mapping HTTP Status Code and Status Message

You can find out the HTTP status code and status message of every REST response received by a REST reference binding. You can then map the HTTP status code and status message to a context parameter. This is helpful when you intend to make decisions based on the HTTP status code of the response received.

Configure the following:

- Supported Type: Basic
- Supported Header Source: HTTP_HEADERS
- Direction: Output
- Header Name: statusCode (For HTTP Status Code)/statusMsg (For HTTP Status Message)
- Context Parameter Name: statusCode (For HTTP Status Code)/statusMsg (For HTTP Status Message)

Modifying a REST Resource Configuration File

Procedure

1. Select a REST Binding on a reference.
2. In the Properties View, click on the hyperlink of the REST resource configuration file. This displays the REST Resource Configuration File Editor.
4. Select the Media Type - Standard JSON or XML. The default is Standard JSON.

   This Media Type applies to all operations. You cannot set the Media Type at an operation level. This Media Type applies to both the request and response of each operation, that is, the default value of "Content-Type" and "Accept" header is derived from Media Type.

5. Click to add a resource. In the Details Section, specify the following resource details.

   - Resource Name: resource name must be unique.
   - Resource Path: Resource Path is the path used to access a resource. It is appended to the value of Context Root.

   The URL is a combination of the following: <machine_name>:<port>/<ContextRoot>/<Resource>

   In TIBCO ActiveMatrix, a WSDL is usually created with operations in it. If a REST service hosts this WSDL, each operation name has a 'Path' (or Resource Path) associated with the operation. The 'Path' may be different from the Operation Name. This 'Path' must be mapped to the corresponding 'Resource' in the REST Resource Configuration file on a reference.

   - Path Parameters: To create a path parameter, create a Path variable represented as {path parameter}. For example, Resource path to access a book with Id is "/book/{id}" where {id} is Path Parameter. Path Parameters are added automatically to the table based on the Resource Path value. In the table, you can edit the data type but you cannot add or remove parameters. Path Parameters cannot be null or empty.

6. Click to add an operation to a resource. In the Details Section, specify operation details such as Operation Name and HTTP Method. Operation Name must be unique across all resources. You
can select **GET**, **PUT**, **POST**, or **DELETE** as the **HTTP Method**. The default is **GET**. For each operation, specify the following request and response details, as appropriate.

<table>
<thead>
<tr>
<th>Request or Response</th>
<th>Type of Request or Response</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Request</strong></td>
<td>Standard JSON Request</td>
<td>In the <strong>Rest Resources</strong> section, select <strong>Request</strong> under <strong>Operation</strong>. In the <strong>Details Section</strong>, click ![+] to add the query parameters or header. In the <strong>Body (JSON)</strong> section, provide the JSON payload details.</td>
</tr>
</tbody>
</table>
|                     | XML Request                 | In the **Rest Resources** section, select **Request** under **Operation**. In the **Details Section**, click ![+] to add the query parameters or header. In the **Body (XML)** section, select the XSD element by clicking **not set** hyperlink or by clicking ![...]. **NOTE**: All the XSD files needed for the configuration of the RRC must be placed in the **Service Descriptors** folder. This includes all imported XSDs from within an XSD.

XSD Element picker only shows XSD Element and not XSD Types. |
| **Response**        | Standard JSON Response      | In the **Rest Resources** section, select **Response** under **Operation**. In the **Details Section**, specify the details in the **Body (JSON)** section or select the **JSON** file using the file picker. |

A request configuration includes query parameters, header, and body. Query parameter and header name cannot be null or empty. Header name cannot be "Accept" or "Content-Type".

**NOTE**: The **Body (JSON)** or **Body (XML)** sections are not displayed if **HTTP Method** is set to **GET** or **DELETE**.
<table>
<thead>
<tr>
<th>Request or Response</th>
<th>Type of Request or Response</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XML Response</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In the <strong>Rest Resources</strong> section, select <strong>Response under Operation</strong>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In the <strong>Details Section</strong>, select the XSD in the <strong>Body (XML)</strong> section.</td>
</tr>
</tbody>
</table>

7. Click 🕒 to add an error type to an operation.
   One operation can have multiple error types. Every error type must be associated with either a single or a list of HTTP Status Codes. If a list of status codes needs to be specified, separate them with a comma.

8. To generate a WSDL file from the `.rrc` file, select the `.rrc` file in the Project Explorer window, and then select **Generate WSDL** from the shortcut menu.
   A WSDL along with XSD (in case of Standard JSON) is generated.

   On the REST reference side, you must use only the WSDL generated from the REST Resource Configuration File Editor to configure the reference. Do not manually edit a WSDL generated from the REST Resource Configuration File Editor. Make sure that the WSDL and the `.rrc` file are always in sync with each other. That is, if you make changes in the REST Resource Configuration File Editor, always regenerate the WSDL file.

**Policies Supported**

TIBCO ActiveMatrix binding Type for REST supports the following policies. For additional information on these policies, refer to the TIBCO ActiveMatrix® Service Grid or TIBCO ActiveMatrix® Service Bus documentation.

**Service Side**

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Authentication</td>
<td>Basic Authentication is a security policy that ensures that a consumer request is validated based on the credentials in the header.</td>
</tr>
</tbody>
</table>

**Reference Side**

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Credential Mapping</td>
<td>Basic Credential Mapping is a policy to ensure that the credentials in the consumer request are validated once and propagated across domains. Credentials are mapped using a password identity provider.</td>
</tr>
</tbody>
</table>
Sample Projects

TIBCO ActiveMatrix Binding Type for REST installation includes sample programs which demonstrates use of the HTTP operations and mediation component.

Service Side

TIBCO ActiveMatrix Binding Type for REST includes the following sample programs for the service side in the TIBCO_HOME\amx\3.3\samples\rest directory.

- **Bookstore sample**  Implemented in Java. Exposes a potential interaction of a bookstore administrator with the bookstore inventory. The sample includes two HTTP GET operations, getBookList and getBookByTitle, and one HTTP POST operation, addBook.

- **Multiplecomplexitytypes sample** Demonstrates the use of a Mediation component to expose a WSDL operation with multiple parts of complex type. Your application might need to perform such mediation because the Binding Type for REST only supports WSDL operations with a single part of complex type.

Reference Side

TIBCO ActiveMatrix Binding Type for REST includes the following sample programs for the reference side in the TIBCO_HOME\amx\3.3\samples\rest directory.

- **Bookstore client sample**  This sample is configured with REST Binding Type on reference with XML media-type. It consumes the 'bookstore' sample service shipped with the product.

- **Facebook client sample**  This sample is configured with REST Binding Type on reference with Standard JSON as the media-type. It invokes an external FaceBook service.

- **Pass-Through Mode Sample** This sample is configured with REST Binding Type on reference to demonstrate the Pass Through Mode.

Executing the Bookstore Sample

Deploy the Bookstore sample using TIBCO Business Studio, review the composite configuration and then run the sample to understand about HTTP operations.

Prerequisites

- Before running the samples, ensure that all the required software has been installed and is operating correctly.

  If you install only the Administration profile and not the SOA Development profile, the samples are not included in the installation.

- Download a REST client such as:
  - POSTMAN REST client: https://www.getpostman.com/postman
  - GitHub RESTClient: https://github.com/wiztools/rest-client/

  The REST client offers an easy to use interface for setting HTTP headers and a simple text box for sending payload in the HTTP body. If you do not download a REST client, you can see results of GET operations in a Web browser, but you cannot perform HTTP POST operations.

  The service operations are implemented in Java, and the service interface is defined in WSDL.

  Load the project in TIBCO Business Studio in order to run the sample.
The bookstore sample illustrates how a bookstore administrator might look up the inventory and add new books. Lookup is by title, or the administrator can get a list of books. Lookup can happen from a Web browser or a REST client. Adding new books can be done from a REST client.

**Importing the Bookstore Sample Project**

Load the project in TIBCO Business Studio to run the sample.

**Procedure**

1. Start TIBCO Business Studio.
2. From the File menu, select **Import**.
3. In the Import dialog, select **General > Existing Projects into Workspace** and click **Next**.
4. Select the root directory of the sample project. Check the **Copy projects into workspace** check box.
5. Click **Finish**.

**Reviewing the WSDL that Defines the Service interface**

Explore the WSDL properties to understand input and output operations for the component service.

**Procedure**

1. In the Project Explorer, select the bookstore project (com.tibco.restbt.sample.bookstore) and open **Composites**.
2. Traverse the hierarchy to get to the BookstoreResource component service, which displays in the Properties tab.
3. Click the WSDL link to display the WSDL in the modeling pane, and explore the inputs and outputs for the different operations.
Reviewing the Composite Configuration

Explore the URL and operations associated with the REST bindings.

The composite includes the Java component and the composite service on which the REST bindings are defined.

Procedure

1. In the design window, select the **BookstoreResource** icon.
2. In the **Promoted Service** pane, examine **RESTService_Binding1** and the operations that are associated with it.

![Promoted Service pane](image)

3. Select **RESTService_Binding1** to examine the HTTP Connector and context root associated with the binding. The two together form the URL that the service uses.

4. Examine the operation. Each operation maps to a WSDL implemented in Java. The **Path** field shows the URL where the operation is exposed.

Running the Bookstore Sample

Start the application and test the HTTP GET and POST operations. You can run the sample from TIBCO BusinessStudio.

Procedure

1. Right click the design panel background and select **Debug in RAD** to start the application.
2. Use an HTTP client such as a Web browser or a the REST client tool to invoke one of the HTTP GET method.

   For example, to test **getBooklist**, specify the following URL in a Web browser or a REST client.
   http://host:port/bookstore/books

   Here:

   - **host** and **port** are required by HTTP. You can find the port by choosing the bug icon in the title bar and selecting **Debug Configuration**. You see that information only after you have run the sample once.
   - **/bookstore** is the context root for **RESTService_Binding1**.
   - **/books** is the ‘Path’ of **getBookList**.

   For samples of calling each method, see the **sample_payloads.txt** file.
3. To test the HTTP POST operation **addBook**, you need a REST client. You can specify the XML code for the book you want to add in the POST request.
Executing the MultipleComplexTypes Sample

Deploy and run the MultipleComplexTypes project to understand use of mediation component for operations with complex types.

TIBCO ActiveMatrix Binding Type for REST supports operations with complex types, but does not allow more than one complex type per operation. At times, you might have a service that is implemented in Java, which has operations with multipart messages with complex types. If you want to make that information available to a REST client, you can use a mediation component. The MultipleComplexTypes sample illustrates this.

Importing the MultipleComplexTypes Sample Project

Load the project in TIBCO Business Studio to run the sample.

Procedure

1. Start TIBCO Business Studio.
2. From the File menu, select Import.
3. In the Import dialog, select General > Existing Projects into Workspace and click Next.
4. Select the root directory of the sample project. Check the Copy projects into workspace check box.
5. Click Finish.

Reviewing the Mediation Flow

Explore the mediation flow to see how multiple parts are mapped to a single field.

Procedure

1. In the Project Explorer, select the multiplecomplextype composite.
2. Examine the SingleToMultipe mediation component in the design window.
3. Double-click the pre-defined mediation flow to launch the Mediation Flow Editor.
4. Double-click the addCustomer operation to display the mapping properties in the Input tab of the Transform Mediation Task panel. You can see that the source has multiple parts (Name, Address, ContactInfo) while mediation context maps the information to a single CustomerInput field.
Running the MultipleComplexTypes Sample
Start the sample project using TIBCO BusinessStudio and run it using a REST client.

Procedure

1. In TIBCO Business Studio, click the design panel background and select Debug in RAD.
2. Open a REST client and set the host and port to match your system and the port on which the service runs.
3. Set the application type to JSON.
4. Select the Body tab and enter one of the sample body strings from the sample_payloads.txt file.

Executing the Bookstore Client Sample (Reference)

Procedure

1. Deploy the Bookstore client sample using TIBCO Business Studio.
2. Review the composite configuration.
3. Run the sample to understand HTTP operations.

Importing the Bookstore Client Sample Project
Load the project in TIBCO Business Studio to run the sample.

Procedure

1. Start TIBCO Business Studio.
2. From the File menu, select Import.
3. In the Import dialog, select General > Existing Projects into Workspace and click Next.
4. Select the root directory of the sample project. Check the Copy projects into workspace check box.
5. Click Finish.
Reviewing the REST Resource Configuration File That Defines the REST Service Interface

**Procedure**

1. In the Project Explorer, select the bookstore client project and open Composites.
2. Traverse the hierarchy to get to the BookstoreResource component reference, which displays in the Properties tab.
3. Select RESTReference_Binding1 and click the REST Resource Configuration File link.
4. Review the REST resource configuration file.

Running the Bookstore Client Sample

Start the application and test the HTTP GET and POST operations. You can run the sample from TIBCO BusinessStudio.

**Procedure**

1. Right click the design panel background.
2. Select Debug in RAD to start the application.

Executing the Facebook Client Sample (Reference)

**Procedure**

1. Deploy the Facebook client sample using TIBCO Business Studio.
2. Review the composite configuration.
3. Run the sample to understand HTTP operations.

Importing the Facebook Client Sample Project

Load the project in TIBCO Business Studio.

**Procedure**

1. Start TIBCO Business Studio.
2. From the File menu, select Import.
3. In the Import dialog, select General > Existing Projects into Workspace and click Next.
4. Select the root directory of the sample project. Check the Copy projects into workspace check box.
5. Click Finish.
Running the Facebook Client Sample
Start the application and test the HTTP GET and POST operations. You can run the sample from TIBCO BusinessStudio.

Procedure

1. Deploy the DAA located in Deployment Artifacts:
   - Facebook Graph APIs are SSL protected and hence the client must be SSL-enabled. The certificate file (.crt) and keystore (.jks) file are packaged along with this sample.
   - The above keystore is referred by Resource Template/FacebookKeystoreProviderResource.cred. Update it with the absolute path to the keystore.
   If the keystore has expired, download it from the Facebook site.
2. Right click the design panel background.
3. Select Debug in RAD to start the application.
4. Use URL http://<Host>:<Port>/fBGraphService?wsdl to load the WSDL in SOAP UI or in the WebService Explorer.
   You will see an operation getUserProfile which accepts two arguments - user and access_token. The access_token argument is the Oauth token required by Facebook REST API and user is the user name or id or me (currently logged in user).
5. Get an Access Token:
   a) Open Facebook Graph Explorer using the following URL: https://developers.facebook.com/tools/explorer/?method=GET&path=me.
   b) After authentication, click Get Acces Token on the right side in Graph API Explorer.
   c) After you get the Access Token, use it while invoking getUserProfile operation.

Executing the Pass-Through Mode Sample (Reference)

Procedure

1. Import the Pass Through Model sample project in TIBCO Business Studio.
   a) Start TIBCO Business Studio.
   b) From the File menu, select Import.
   c) In the Import dialog, select General > Existing Projects into Workspace and click Next.
   d) Select the root directory of the sample project. Check the Copy projects into workspace check box.
   e) Click Finish.
2. Review the composite configuration.
3. Deploy the DAA in TIBCO ActiveMatrix Administrator.
4. Test the HTTP operations.

Executing the rest.context Sample

The rest.context sample includes:
- REST SOA projects (/rest-soap-projects/):
- /com.tibco.restbt.context.sample.restjava: Sample Java IT Provider Implementation for REST-Java App (1)
- /com.tibco.restbt.context.sample.restjavarest : Sample JavaIT Provider/Consumer Implementation for REST-Java-REST App (3)

- SoapUI projects to invoke the REST Apps (/SoapUIprojects/):
  - context-REST-Java-REST-soapui-project: SOAPUI project to send a REST Request to the REST-Java-REST App (3) with the HTTP Headers
  - context-REST-Java-soapui-project.xml: SOAPUI project to send a REST Request to the REST-Java App (1) with the HTTP Headers

- SoapUI projects to invoke the REST Apps (/SoapUI-projects/):
  - context-REST-Java-REST-soapui-project: SOAPUI project to send a REST Request to the REST-Java-REST App (3) with the HTTP Headers
  - context-REST-Java-soapui-project.xml: SOAPUI project to send a REST Request to the REST-Java App (1) with the HTTP Headers

**Running the rest.context Example**

1. Launch TIBCO ActiveMatrix Administrator.
2. Create a HTTP Connector Resource Template httpConnector with host as localhost and port as 9897.
3. Create and Install corresponding HTTP Connector Resource Instance httpConnector on Runtime Node.
4. For REST-Java App, deploy the following DAA:
   com.tibco.restbt.context.sample.soa/Deployment Artifacts/restjava.daa
5. For REST-Java-REST App, deploy the following DAA:
   com.tibco.restbt.context.sample.soa/Deployment Artifacts/restjavarest.daa
6. To send a REST Request to the REST-Java App, launch the SOAPUI project context-REST-Java-soapui-project.xml using SoapUI 5.0.0, and send the following REST Request:
7. To send a REST Request to the REST-Java-REST App, launch the SOAPUI project context-REST-Java-REST-soapui-project.xml using SoapUI 5.0.0, and send the following REST Request:

```xml
GET BookstoreResource [/BookstoreResource/getBookList Request 1]
```

Breakdown of the rest.context Scenario

When a REST Request is sent to the REST-Java-REST App with the HTTP Headers (for example, `bookName` and `bookID`) using the SOAPUI project context-REST-Java-REST-soapui-project.xml:

1. they are mapped to context parameters that can be retrieved in the Java IT Implementation (“Java1 (service-inbound)” in the Runtime Node logs shown later)
2. new basic header values and a collection of headers is sent to the REST-Java App via context parameters (“Java1 (reference-inbound)” in the Runtime Node logs shown later)
3. the headers are retrieved in the REST-Java App’s Java IT implementation (“Java2 (service-inbound)” in the Runtime Node logs shown later)
4. new outbound headers are set in the REST-Java App’s Java IT implementation (basic and bag) to be sent back to the REST-Java-REST App (Java2 (service-outbound) in the Runtime Node logs shown later)
5. the headers sent by REST-Java App are retrieved in REST-Java-REST App’s Java IT implementation (“Java1 reference-outbound)” in the Runtime Node logs shown later)
6. some headers are set in REST-Java-REST App’s Java IT implementation to be sent back to the originating client (“Java1 (service-outbound)” or “Java1 (service-fault)” in the Runtime Node logs shown later, depending on the value of “storename” (“success” and “fault” respectively))

REST-Java-REST: Success Scenario of REST-Java
Runtime Node Logs

[INFO ] [rest-java-rest] stdout - storename = success
[INFO ] [rest-java-rest] stdout - storename = success
[INFO ] [rest-java-rest] stdout - Getting requestHeadersAllCP = {Host=localhost:9897, User-Agent=Apache-Http-Client/4.1.1 (java 1.5), bookID=1602, Connection=Keep-Alive, bookName=The Kite Runner}
[INFO ] [rest-java-rest] stdout - storename = success
[INFO ] [rest-java-rest] stdout - Getting requestHeadersAllCP = {bookPublishYear=1960, bookAuthor=Harper Lee}
[INFO ] [rest-java-rest] stdout - storename = success
[INFO ] [rest-java-rest] stdout - Getting requestHeadersAllCP = {bookPublishYear=1960, bookAuthor=Harper Lee, Cookie=JSESSIONID=1wt523hzwhpvlgyxt8frpsu, Cookie2= $Version=1, Host=localhost:9897, Accept-Charset=UTF-8, BookID=1501, Connection=Keep-Alive, Accept=application/json, ContentType=application/json; charset=UTF-8, bookName=How to Kill a Mockingbird}
[INFO ] [rest-java] stdout - storename = success
[INFO ] [rest-java] stdout - storename = success
[INFO ] [rest-java] stdout - storename = success
[INFO ] [rest-java] stdout - Getting bookCategory = Classic
[INFO ] [rest-java] stdout - Getting bookQuantity = 50
[INFO ] [rest-java-rest] stdout - storename = success
[INFO ] [rest-java-rest] stdout - storename = success
[INFO ] [rest-java-rest] stdout - storename = success
[INFO ] [rest-java-rest] stdout - Getting bookCategory = Classic
[INFO ] [rest-java-rest] stdout - Getting bookQuantity = 50
[INFO ] [rest-java-rest] stdout - Setting responseHeadersAll: = {bookPublishYear=1960, bookAuthor=Harper Lee}
.Runtime Node Logs

```
[INFO ] [rest-java-rest] stdout - ------------ Java1 (service-inbound) ----------
[INFO ] [rest-java-rest] stdout - storename = fault
[INFO ] [rest-java-rest] stdout - ------------ Java1 (service-inbound) ----------
[INFO ] [rest-java-rest] stdout - Getting bookName = The Kite Runner
[INFO ] [rest-java-rest] stdout - Getting bookID = 1602
[INFO ] [rest-java-rest] stdout - Getting requestHeadersAllCP = {Host=localhost:9897, User-Agent=Apache-Http-Client/4.1.1 (java 1.5), bookID=1602, Connection=Keep-Alive, bookName=The Kite Runner}
[INFO ] [rest-java-rest] stdout - ------------ Java1 (service-inbound) ----------
[INFO ] [rest-java-rest] stdout - Setting bookName = How to Kill a Mockingbird
[INFO ] [rest-java-rest] stdout - Setting bookID = 1501
[INFO ] [rest-java-rest] stdout - Setting requestHeadersAllCP = {bookPublishYear=1960, bookAuthor=Harper Lee}[
[INFO ] [rest-java-rest] stdout - storename = fault
[INFO ] [rest-java-rest] stdout - ------------ Java2 (service-inbound) ----------
[INFO ] [rest-java-rest] stdout - Getting bookName = How to Kill a Mockingbird
[INFO ] [rest-java-rest] stdout - Getting bookID = 1501
[INFO ] [rest-java-rest] stdout - Getting requestHeadersAllCP = {Host=localhost:9897, Accept-Charset=UTF-8, Accept=application/json, Content-Type=application/json; charset=UTF-8, bookId=1501, bookName=How to Kill a Mockingbird}
[INFO ] [rest-java-rest] stdout - ------------ Java2 (service-inbound) ----------
[INFO ] [rest-java-rest] stdout - Setting bookCategory = Classic
[INFO ] [rest-java-rest] stdout - Setting bookQuantity = 50
[INFO ] [rest-java-rest] stdout - Setting responseHeadersAll: = {bookPublishYear=1960, bookAuthor=Harper Lee}
[INFO ] [rest-java-rest] stdout - ------------ Java1 (service-outbound) ----------
[INFO ] [rest-java-rest] stdout - Getting bookCategory = Classic
[INFO ] [rest-java-rest] stdout - Getting bookQuantity = 50
[INFO ] [rest-java-rest] stdout - Getting responseHeadersAll: = {bookPublishYear=1960, bookAuthor=Harper Lee, bookCategory=Classic, bookQuantity=50, Date=Thu, 08 Oct 2015 21:03:43 GMT, Content-Length= 252, Expires=Thu, 01 Jan 1970 00:00:00 GMT, Set-Cookie=JSESSIONID=1wt523hzhpvlgyxt8rpsu; Path=/BookStoreBackend; HttpOnly, Content-Type=application/json; charset=UTF-8}
```
Executing the rest.extendedJSONConversion Sample

The rest.extendedJSONConversion sample includes:

- **REST SOA projects (/rest-soa-projects/):**
  - restbt.sample.extended.json.conversion/Composites/restbt.sample.extended.json.conversion.composite
  - (restbt.sample.extended.json.conversion/DeploymentArtifacts/restbt.sample.extended.json.conversion.daa): REST-Java (Provider) and REST-Java-REST (Client) components to demonstrate the enhancements
  - com.sample.restbt.sample.extended.json.conversion: Provider and Client Java Implementations

- **SoapUI projects to invoke the REST Apps (/SoapUI-projects/REST-ExtendedJSONConversion-soapui-project.xml):**
  - Service “InvokeRESTProvider”: SOAPUI project to send a REST request to the REST Provider (REST-Java)
  - Service “InvokeRESTClient”: SOAPUI project to send a REST request to the REST Client (REST-Java-REST) which in turns sends a REST request to the REST Provider (REST-Java)

The SOAPUI projects will work as-is if the REST Service Bindings (that use HTTP Connector “httpConnector”) are deployed on localhost:9897.

**Provider side (Service/Outbound/Response): XML-to-JSOn Conversion of String XSD Element**

Consider an XSD Element that is defined as a “string” in the Provider-side WSDL Schema (identified by the 'type' attribute, which must be set to XSD string, for example 'type="xsd:string"' where "xsd" points to the XML Namespace “http://www.w3.org/2001/XMLSchema”).

If the XML Response contains a non-numeric String as the value for that XSD Element (for example "teststring" or "test1234"), the JSON Response object is serialized correctly as part of the standard XML-to-JSOn Conversion that is as a JSON String (“teststring” or “test1234”).

However, if the XML Response contains a purely numeric String as the value for that XSD Element (for example "1234"), then the standard XML-to-JSOn Conversion converts the element to a JSON Number (for example 1234 as opposed to "1234" that is not a JSON String), which would result in an inconsistent conversion.

This holds true for other JSON types as well for example Double, Boolean, and so on. With this release, the XSD Element's (in the sample scenario, the “Value” element in Sample.wsdl) "type" attribute will be factored into the XML-to-JSOn Conversion specifically for JSON Strings that is if the XSD Element is of type "string", the resultant JSON Object will be a JSON String as well and not a JSON Number.

This affects the Service-side RESPONSE. The REQUEST is not affected.

**Client side (Reference/Inbound/Request): XML-to-JSOn Conversion of String XSD Element**

Whether or not an element is of type “string” is inferred from the Reference-side REST Resource Configuration (RRC) file, specifically from the “Request” of the “POST” method.

In the sample scenario, that would the “Request” of the “addOperation”. If you indicate via the RRC file that “Value” is a string (by use of "Value":"123"”), then with this release, the POST Request generation will honor the “string” type by sending a JSON String, even if the “Value” contains only a purely numeric value.

This affects the POST method’s REQUEST. The POST method’s RESPONSE is not affected.
This behavior is demonstrated in the “Test Scenario” section below.

Only inline schemas are supported on the Service-side that is the XSD elements (of type string) that are of interest to this sample must be present in the Service-side WSDL.

Running the rest.extendedJSONConversion Example

1. Launch TIBCO ActiveMatrix Administrator.
2. Set the TRA property `com.tibco.amf.runtime.bindingtype.rest.extendedJsonConversion` to `true` for the appropriate Runtime Node (via Administrator UI or by adding the line `java.property.com.tibco.amf.runtime.bindingtype.rest.extendedJsonConversion=true` in the .tra file of the Runtime Node) and restart the Runtime Node.
3. Enable the `com.tibco.amx.bt.rest` logger in DEBUG level on the Runtime Node.
4. Create a HTTP Connector Resource Template `httpConnector` with `host` as localhost and `port` as 9897.
5. Create and Install corresponding HTTP Connector Resource Instance “httpConnector” on Runtime Node.
6. For both REST-Java and REST-Java-REST Apps, deploy the following DAA:

   `/rest-soa-projects/restbt.sample.extended.json.conversion/DeploymentArtifacts/restbt.sample.extended.json.conversion.daa`

7. To send a REST Request to the REST-Java (Provider) App, launch the SOAPUI project REST-ExtendedJSONConversion-soapui-project.xml using SoapUI 5.0.0, and use the “InvokeRESTProvider” service. This will send a REST request to the REST-Java app and will demonstrate the Provider-side (Service) behavior.

   Set HTTP Request Header to Accept = application/json.

   GET method’s RESPONSE (returnOne)
   Request URL: http://localhost:9897/Sample/returnOne (HTTP method: GET)
   Sample Response (Standard JSON)

   
   ```
   {"returnOneResponse": {"Flights": [{
   "Arrival": "arrival",
   "Departure": "123"
   }]}},
   ```

   Note the [ ] indicating that Flights is an Array, based on the schema. Also, “123” is a string, which is schema-compliant.

8. To send a REST Request to the REST-Java-REST (Client) App, launch the SOAPUI project REST-ExtendedJSONConversion-soapui-project.xml using SoapUI 5.0.0, and use the “InvokeRESTClient” service. This will send a REST request to the REST-Java-REST app, which in turn will send a rest to the REST-Java app. This will demonstrate the Client-side (Reference) behavior.

   POST method’s REQUEST (addOperation)
   Request URL: http://localhost:9897/SampleClient/addOperation (HTTP method: POST)
   Set HTTP Request Header:
   Accept = application/json and Content-Type = application/json

   SOAPUI Request to first REST Service Endpoint:

   ```
   {"addOperation": {"Arrays": [{"Key": "ABC", "Value": "123"}]}}
   ```

   POST Request generated by REST Reference Endpoint:

   ```
   {"Arrays": [{"Key": "ABC", "Value": "123"}]}
   ```
Note the [] indicating that Arrays is an Array, based on the schema. Also, “123” is a string, based on the schema.

Response (Standard JSON):

```json
{"addOperationResponse": {"Arrays": [{
"Key": "ABC",
"Value": "123"
}]}}
```

Note the [] indicating that Arrays is an Array, based on the schema. Also, “123” is a string, based on the schema.

**REST-Java-REST: Runtime Node Logs for Rest-Java**

25 Jan 2016 17:23:23,168 [httpConnector_2] [DEBUG] []
PromotedServiceName=SampleClient/Sample, BindingName=RESTService_Binding1, BindingType=REST, URL=http://localhost:9897/SampleClient/addOperation, ComponentURI=urn:amx:DevEnvironment/restbt.sample.extended.json.conversion/JavaClient_1.0.0.v2016-01-25- 1703_inbound_service_SampleClient/ Sample RESTService_Binding1, OperationName={http://HdrTest/Sample/addOperation, CorrelationID=led56457-33c4-47e0-aff4-e91b03b4cb92, ContextID=led56457-33c4-47e0- aff4-e91b03b4cb92, ParentContextID=null, RequestHeaders={RequestAttributes : amx.connector.name = httpConnector; org.mortbay.jetty.newSessionId = hxej3j3zzt0wde68d04z9x4d; component.URI = urn:amx:DevEnvironment/ restbt.sample.extended.json.conversion#servicebinding(SampleClient/ RESTService_Binding1)_1.0.0.v2016-01-25-1703; } {RequestParameters : } { Protocol = HTTP/1.1 } { RequestURI = /SampleClient/addOperation } { Method = POST } { HTTPHeader : Content-Type = application/json; Content-Length = 58; Host = localhost:9897; Connection = Keep-Alive; User-Agent = Apache-HttpClient/4.1.1 (java 1.5); } { QueryString : null }{PathInfo : /addOperation }

25 Jan 2016 17:23:23,255 [httpConnector_2] [DEBUG] []
com.tibco.amx.bt.rest.RESTHttpClient - Request Message from REST Promoted Reference to External Service.
PromotedReferenceName=extendedConversionRrcResource, BindingName=RESTReference_Binding1, BindingType=REST, URL=http://localhost:9897/Sample/addOperation, ComponentURI=urn:amx:DevEnvironment/ restbt.sample.extended.json.conversion#servicebinding(extendedConversionRrcResource/ RESTService_Binding1)_1.0.0.v2016-01-25-1703; } {RequestParameters : } { Protocol = HTTP/1.1 } { RequestURI = /Sample/addOperation } { Method = POST } { HTTPHeader : Content-Type = application/json; Accept: application/json, Accept-Charset: UTF-8; Content-Length = 40; Host = localhost:9897; Connection = Keep-Alive; User-Agent = Apache-HttpClient/4.1.1 (java 1.5); } { QueryString : null }{PathInfo : /addOperation }

PromotedServiceName=SampleClient/Sample, BindingName=RESTService_Binding1, BindingType=REST, URL=http://localhost:9897/SampleClient/addOperation, ComponentURI=urn:amx:DevEnvironment/restbt.sample.extended.json.conversion/JavaProvider_1.0.0.v2016-01-25-1703_inbound_service_SampleClient/ Sample RESTService_Binding1, OperationName={http://HdrTest/Sample/addOperation, CorrelationID=46080a36-a928-46db-be47-2da6cb475b16, ContextID=46080a36-a928-46db-be47-2da6cb475b16, ParentContextID=null, RequestHeaders={RequestAttributes : amx.connector.name = httpConnector; org.mortbay.jetty.newSessionId = lutct7734eussb1od1b0sfpx60; component.URI = urn:amx:DevEnvironment/ restbt.sample.extended.json.conversion#servicebinding(SampleClient/ RESTService_Binding1)_1.0.0.v2016-01-25-1703; } {RequestParameters : } { Protocol = HTTP/1.1 } { RequestURI = /Sample/addOperation } { Method = POST } { HTTPHeader : Content-Type = application/json; charset=UTF-8, Accept: application/json, Accept-Charset: UTF-8, Content-Length = 40; Host = localhost:9897; Connection = Keep-Alive; }{ QueryString : null }{ PathInfo : /addOperation }

TIBCO ActiveMatrix® Service Grid REST Binding Development Guide
Response (Reply) Message from REST Promoted Service to External Consumer.
PromotedServiceName=Sample/Sample, BindingName=RESTService_Binding1, BindingType=REST, URL=http://localhost:9897/Sample/addOperation,
ComponentURI=urn:amx:DevEnvironment/restbt.sample.extended.json.conversion/JavaProvider_1.0.0.v2016-01-25-1703_inbound_service_Sample/
PromotedReferenceName=extendedConversionRrcResource, BindingName=RESTReference_Binding1, BindingType=REST, URL=N/A,
ComponentURI=urn:amx:DevEnvironment/restbt.sample.extended.json.conversion/JavaClient_1.0.0.v2016-01-25-1703_outbound_reference_extendedConversionRrcResource_RESTReference_Binding1,
PromotedServiceName=SampleClient/Sample, BindingName=RESTService_Binding1, BindingType=REST, URL=http://localhost:9897/SampleClient/addOperation,
ComponentURI=urn:amx:DevEnvironment/restbt.sample.extended.json.conversion/JavaClient_1.0.0.v2016-01-25-1703_inbound_service_SampleClient/Sample_RESTService_Binding1,
**Limitations**

This section lists the limitations and usage guidelines.

### General Limitations

#### Trailing Slash in Request URL

- A REST BT service does not differentiate between a URL ending with a trailing slash (/) and one without a trailing slash.
- A REST BT reference differentiates between a URL ending with a trailing slash and a URL not ending with a trailing slash. The reference treats them as different URLs and depends on how you have configured the resource path in the resource configuration file. For example:
  - If the resource path is books/ (with a trailing /), the resulting URL is `http://host:port/Context-Root/books/`.
  - If the resource path is books, the resulting URL is `http://host:port/Context-Root/books`.
  - If the resource path is books/ and a minimum of one query parameter (for example, ID), the resulting URL is `http://host:port/Context-Root/books?ID=value`.

#### Numbers and Special Characters

- Numbers and special characters (period, comma, hyphen, or hash) are not supported in a Standard JSON payload key.

### Validation Limitations

#### Design-time

For Standard JSON, if you configure the resource configuration file from payloads that do not have a well-defined root-level element, you must map the response and faults body to a unique status code.

### Service Limitations

You cannot override the default status code (200) returned from a REST service.

#### Query and Path Parameters

- Mixing of path parameters and path parts in a resource URI is not supported. For example, an online bookstore has a service to update (add/delete) the number of books in the inventory based on ISBN. The resource URI of the update operations is:
  


  Such resource URIs are not supported. However, the current implementation of REST BT supports the resource URI where all the path parameters are suffixed. For example:


- Usage guidelines for query and path parameters:
  
  - Path parameters and query parameters must be of `Simple` type.
- Query parameters are optional, if not specified.
- Configuring a custom default value for a query parameter is not supported.
- The key of the query parameter is the WSDL part name.
- Path parameters and query parameters must map to a WSDL part. The implementation does not receive query parameters that are not mapped to any WSDL part name.

Request Response

- REST Request body - There can be at the most one WSDL part referring to a Complex-Type element, which if present is considered as HTTP body.
- REST Request/Response - The WSDL part representing the REST body, must refer to Element and not the Type. Simple type in body is not supported. However, you can construct a Simple type as Complex-Type Element in a WSDL.

Schema Limitations

Standard JSON

- Standard JSON is not namespace-aware. All the elements, types, and attributes must be in a single namespace. No foreign namespace elements or attributes are supported.
- Global attributes are not supported in the schema, for a REST service with Standard JSON as the media type.
- For Standard JSON, the elementFormDefault attribute must always be set to qualified.
- To pass an attribute using JSON, you must represent it as "@Key": "Value". That is, prefix the key with "@". For example, the following payload works:

```json
{
  "PersonDetailsInputNestedElements": {
    "PersonDetailsInputElementComplexType": {
      "@PersonDOB": "2001-10-26T21:32:52",
      "@PersonName": "Mr Person"
    },
    "FriendDetailsInputElementComplexType": {
      "@PersonDOB": "2001-10-26T21:32:52",
      "@PersonName": "Mr Friend"
    }
  }
}
```

This convention is not specified by Standard JSON; it is specific to the TIBCO implementation.
## Troubleshooting

<table>
<thead>
<tr>
<th>Issue</th>
<th>Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>For a REST BT Reference, the RRC file shows error messages even when the validation shows no errors in the file. Error messages similar to the following are displayed in the Problems window:</td>
<td>Clean the project.</td>
</tr>
<tr>
<td>'NewRrcResource1.rrc' has some error. Please validate manually.</td>
<td></td>
</tr>
<tr>
<td>REST_Binding: 'NewRrcResource1.rrc' has some error. Please validate manually.</td>
<td></td>
</tr>
<tr>
<td>For a REST BT Reference, the RRC to WSDL generation does not generate an array type in XSD if the JSON payload has an array represented by [ ]. As an example, for the following payload:</td>
<td>Make sure the array has at least two elements.</td>
</tr>
<tr>
<td>{getMeresponseType: {</td>
<td></td>
</tr>
<tr>
<td>&quot;id&quot;: &quot;100006769630394&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;work&quot;: [</td>
<td></td>
</tr>
<tr>
<td>&quot;employer&quot;: {</td>
<td></td>
</tr>
<tr>
<td>&quot;id&quot;: &quot;47358345258&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;name&quot;: &quot;TIBCO Software Inc.&quot;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>The generated XSD must look like:</td>
<td></td>
</tr>
<tr>
<td>&lt;xs:complexType name=&quot;getMeresponseType&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td><a href="">xs:sequence</a></td>
<td></td>
</tr>
<tr>
<td>&lt;xs:element type=&quot;xs:long&quot; name=&quot;id&quot; /&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;xs:element type=&quot;workType&quot; name=&quot;work&quot; maxOccurs=&quot;unbounded&quot; minOccurs=&quot;0&quot; /&gt; //array</td>
<td></td>
</tr>
<tr>
<td>&lt;/xs:sequence&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/xs:complexType&gt;</td>
<td></td>
</tr>
<tr>
<td>If you send a request from a REST Client running on the same machine where REST Service is deployed, you may see REST client IP value as [0:0:0:0:0:0:0:1] on the TIBCO Service Performance Manager dashboard.</td>
<td>Make sure the IPv4 DNS resolution is used in the REST Client setting.</td>
</tr>
</tbody>
</table>