TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON

User's Guide

Software Release 2.1
October 2017

Two-Second Advantage®

TIBCO™

Important Information

Contents

Figures

Tables

Preface

TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON used with TIBCO ActiveMatrix BusinessWorks, allows you to expose TIBCO ActiveMatrix BusinessWorks processes as RESTful web services, invoke RESTful web service APIs, and translate data between JSON and XML formats.

Topics

Related Documentation

This section lists documentation resources you may find useful.

TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON Documentation

The following documents form the TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON documentation set:

- *TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON Installation* Read this manual for instructions on site preparation and installation.
- *TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON User's Guide* Read this manual for instructions on using the product.
- *TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

Other TIBCO Product Documentation

You may find it useful to read the documentation for the following TIBCO products:

- TIBCO ActiveMatrix BusinessWorks™
- TIBCO Administrator™
- TIBCO Designer™
- TIBCO Enterprise Message Service™
- TIBCO Hawk®
- TIBCO Rendezvous®
- TIBCO Runtime Agent™

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
ENV_NAME *TIBCO_HOME*	TIBCO products are installed into an installation environment. A product installed into an installation environment does not access components in other installation environments. Incompatible products and multiple instances of the same product must be installed into different installation environments. An installation environment consists of the following properties: • **Name** Identifies the installation environment. This name is referenced in documentation as *ENV_NAME*. On Microsoft Windows, the name is appended to the name of Windows services created by the installer and is a component of the path to the product shortcut in the Windows Start > All Programs menu. • **Path** The folder into which the product is installed. This folder is referenced in documentation as *TIBCO_HOME*.
RESTJSON_HOME	TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON is installed into a directory within *TIBCO_HOME*. This directory is referenced in documentation as *RESTJSON_HOME*. The default value of *RESTJSON_HOME* depends on the operating system. For example, on Windows systems, the default value is C:\tibco\bw\plugins\restjson.
code font	Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example: Use MyCommand to start the foo process.
bold code font	Bold code font is used in the following ways: • In procedures, to indicate what a user types. For example: Type **admin**. • In large code samples, to indicate the parts of the sample that are of particular interest. • In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, MyCommand is enabled: MyCommand [**enable** | disable]

Table 1 General Typographical Conventions (Cont'd)

Convention	Use
italic font	Italic font is used in the following ways: • To indicate a document title. For example: See *TIBCO ActiveMatrix BusinessWorks Concepts*. • To introduce new terms. For example: A portal page may contain several portlets. *Portlets* are mini-applications that run in a portal. • To indicate a variable in a command or code syntax that you must replace. For example: MyCommand *PathName*
Key combinations	Key name separated by a plus sign indicate keys pressed simultaneously. For example: Ctrl+C. Key names separated by a comma and space indicate keys pressed one after the other. For example: Esc, Ctrl+Q.
	The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.
	The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result.
	The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken.

TIBCO Product Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, or join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website mainly in the HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit https://docs.tibco.com.

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit https://www.tibco.com/services/support.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at https://support.tibco.com.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to https://support.tibco.com. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to https://community.tibco.com.

Chapter 1 Plug-in Introduction

This chapter gives an overview of TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON.

Topics

Product Overview

TIBCO ActiveMatrix BusinessWorks is a standards-based service creation for companies looking to integrate their computing environment and automate their business processes. TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON used with TIBCO ActiveMatrix BusinessWorks, enables you to invoke RESTful web service API, expose TIBCO ActiveMatrix BusinessWorks processes as RESTful web services, and translate data between JSON and XML formats. The main functionalities of the plug-in are as follows:

- Support for exposing TIBCO ActiveMatrix BusinessWorks processes as RESTful web services by binding HTTP requests to TIBCO ActiveMatrix BusinessWorks processes. See REST Dispatch and Reply on page 56 for more information.
- Support for invoking RESTful web service API and receiving responses from Service Providers. See Invoke REST API on page 41 for more information.
- Support for converting data between JSON format and XML format with the following schema types: Generic, XSD, and Java Classes. See Supported Schema Types for Formatting XML Data on page 18 for more information.
- Support for securing RESTful web services call with the following authentication types: Basic Authentication, OAuth 1.0, OAuth 2.0 (client side), and No Authentication. See OAuth1.0 on page 40 and Invoke REST API on page 41 for more information.
- Support for exporting a RESTful web service to a WADL file. See Exporting RESTful Web Services to a WADL File on page 88.
- Support for exporting a RESTful web service to Swagger. See Exporting RESTful Web Service to Swagger on page 89 for more information.
- Support for creating a custom RESTful web service and binding HTTP requests from a custom RESTful web service to TIBCO ActiveMatrix BusinessWorks processes. See Creating a RESTful Web Service on page 79 for more information.
- Support for creating a custom WADL file to describe a RESTful web service. See WADL Palette on page 67 for more information.
- Support for invoking a Service Provider with SSL configurations. See Support for SSL Configuration on page 130 for more information.
- Support for handling cross-origin requests. See Support for Cross-Origin Requests on page 131 for more information.
- Support for loading Swagger JSON files by Swagger tools. See Using the Swagger Tools on page 35 for more information.

- Support for generating XML schemas from JSON data by using JSON tools. See Using the JSON Tools on page 34 for more information.

Chapter 2 Getting Started

This chapter describes the steps to configure and run TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON in TIBCO Designer.

Topics

Overview

TIBCO ActiveMatrix BusinessWorks is a scalable, extensible, and easy-to-use integration platform that allows you to develop and test integration projects. TIBCO ActiveMatrix BusinessWorks includes a graphical user interface, TIBCO Designer, for configuring business processes, and an engine that runs the processes.

For detailed information about how to configure processes, see TIBCO Designer documentation, which can be accessed by selecting **Help > Designer Help** from the menu in the TIBCO Designer window.

A typical configuration and deployment procedure includes the following steps:

1. Starting TIBCO Designer
2. Creating a Project
3. Creating an OAuth1.0 Shared Resource
4. Creating a TIBCO ActiveMatrix BusinessWorks Process
5. Adding Activities to the Process
6. Testing the Process
7. Deploying a Project

Starting TIBCO Designer

TIBCO Designer is used to configure TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON instances. To start TIBCO Designer, run one of the following platform-specific commands:

- On Microsoft Windows

  From the Start menu, select **All Programs > TIBCO > TIBCO Designer *version_number* > TIBCO Designer**

  or

  At the command prompt, type *TIBCO_HOME*\designer*version_number*\bin\designer.exe.

- On UNIX

  Run *TIBCO_HOME*/designer/*version_number*/bin/designer.

After the command runs successfully, the TIBCO Designer Startup panel is displayed.

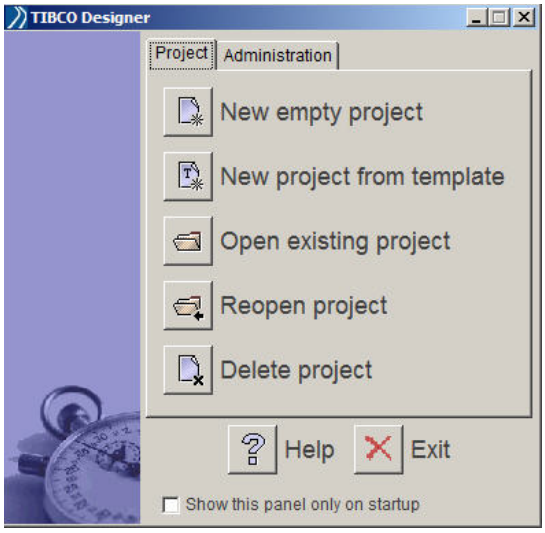
Creating a Project

After starting TIBCO Designer, you can create a project or open an existing project in the displayed Startup panel. A project contains configuration files, which define options used by a runtime plug-in.

To create a project in TIBCO Designer, perform the following steps:

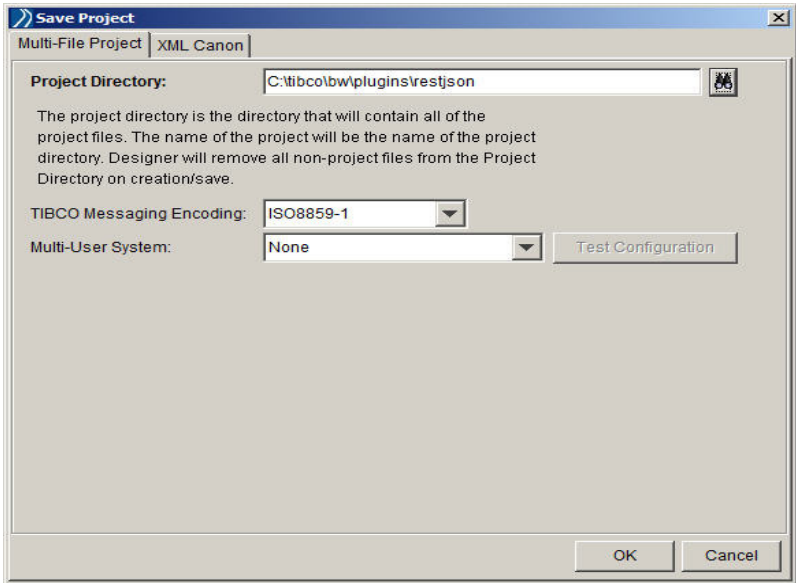
1. In the TIBCO Designer Startup panel, click **New Empty Project** as shown in Figure 1.

Figure 1 Create a New Project



2. In the **Save Project** dialog box, save the project at a specific location.
   a. Click **Select a file from the file system** next to the **Project Directory** field on the **Multi-File Project** tab.
   b. Navigate to the location where you intend to save the project and specify a name for the project as shown in Figure 2. Click **OK**.

Figure 2 Save a New Project



The TIBCO Designer window is displayed with the newly created project.

Creating an OAuth1.0 Shared Resource

To use the OAuth 1.0 authentication, you must create an OAuth1.0 shared resource in an existing project and then run an Invoke REST API process.

To create an OAuth1.0 shared resource, perform the following steps:

1. Open the project folder you created in Creating a Project on page 8 in the Project panel.
2. Expand the **REST and JSON** palette in the Palette panel, and drag the **OAuth1.0** icon to the Design panel to create a shared resource.
3. Configure the created OAuth1.0 shared resource and then click **Apply** to save the configuration.

   For more information about configuring OAuth1.0 shared resources, see OAuth1.0 on page 40.
4. Save the project.

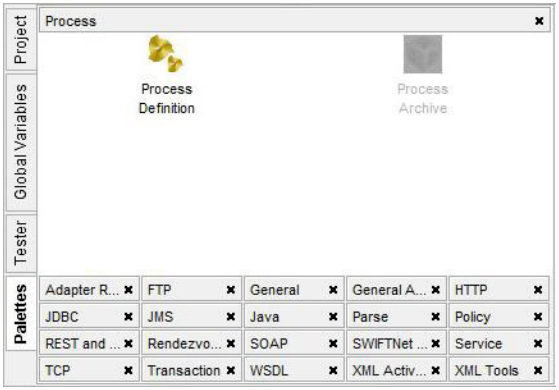
Creating a TIBCO ActiveMatrix BusinessWorks Process

To deal with certain workflows, you need to create TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON processes.

Perform the following steps to create a process:

1. Right-click the project folder you created in Creating a Project on page 8, select **New Folder** item from the menu that is displayed, and rename the folder to Processes.
2. Click the Processes folder, expand the **Process** palette in the Palette panel, and drag the **Process Definition** icon to the Design panel. As a result, the **Process Definition** activity can be seen in the Design panel as shown in Figure 3.

Figure 3 Process Definition



3. Configure the process based on your requirements.

   For more information about how to configure processes, see *TIBCO Designer User's Guide*.
4. Save the project.

Adding Activities to the Process

To add activities to the process, perform the following steps:

1. In TIBCO Designer, open the process created in Creating a TIBCO ActiveMatrix BusinessWorks Process on page 11. By default, the Start and End activities are displayed in the Design panel.
2. From the Palettes panel, select the activities of the TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON to define the process.
   a. Expand the **REST and JSON** palette in the Palettes panel, and drag the activities to the Design panel.
   b. You can also add activities from other palettes to the process. For example, File activities.

If you cannot view the **REST and JSON** palette in the Palettes panel, select **Palettes > Activities > REST and JSON** from the TIBCO Designer menu to make the palette visible.

3. Click **Create transition** in the TIBCO Designer toolbar to draw transitions between activities.
4. Configure each activity in the process.

   See REST and JSON Palette on page 37 for details about REST and JSON activity configurations.

Once the process definition is complete, you can use the test mode tool to test the process. See Testing the Process on page 13 for more details.

Testing the Process

After creating and configuring a process, you can test the process by using the test mode tool. Testing the process helps you check whether the process works properly before deploying it. For more information, see *TIBCO ActiveMatrix BusinessWorks™ Process Design Guide*.

Deploying a Project

Before deploying a project, you must create an Enterprise Archive file (EAR file) that contains the configuration for the process definition. You can upload the archive to TIBCO Administrator to deploy the associated application.

Complete the following steps to deploy a configured project:

1. Save the project in the TIBCO Designer window. Ensure that the processes saved in the project is tested.
2. To create an Enterprise Archive, select **Tools > Create Project EAR** from the TIBCO Designer menu.
3. Expand the newly created Enterprise Archive and select the **Process Archive**. Click  on the **Processes** tab.
4. In the **Select A Resource** dialog box, expand the folder to select the process definition you want to include and then click **Apply**.
5. To build an archive, click the Enterprise Archive that you created in Step 2. Then click **Build Archive** in the **Configuration** tab. An EAR file is generated and saved in the location specified in the **File Location** field in the **Configuration** tab.
6. Deploy the project in TIBCO Administrator. Start TIBCO Administrator and upload the EAR file for the project. Deploy the application and start the process.

   See *TIBCO ActiveMatrix BusinessWorks™ Administration* for details about how to deploy a project in TIBCO Administrator.

Chapter 3 Supported XML and JSON Conversion

This chapter explains the supported normal conversion rules and Badgerfish conversion rules for mapping data between JSON and XML. It also describes the supported schema types for formatting XML data and the supported root node conversion.

Topics

JSON and XML Conversion Overview

The different properties of XML and JSON formats do not guarantee all content in XML is converted to JSON. For example, XML supports inline metadata with tags and attributes, which has no such standard form in JSON. TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON provides the schema types to format XML data and the conversion rules to map data:

- Schema Types
- Conversion Rules

When converting data between JSON and XML, you can format XML data with the supported schema types.

Schema Types

You can use the following schema types to format XML data:

- Generic
- XSD
- Java Classes

See Supported Schema Types for Formatting XML Data on page 18 for more information.

Conversion Rules

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports the following conversion functionalities:

- Converting data with the Badgerfish conversion rule, which is a comprehensive conversion between JSON and XML.

  See Badgerfish Conversion Rule on page 21 for more information.

- Converting data with normal conversion rules, which is a lossy conversion compared to the Badgerfish conversion.

  See Normal Conversion Rule on page 25 for more information.

- Root node control.

  See Root Node Conversion Rule on page 28 for more information about root node conversion.

- Null and empty values control.

  See Null and Empty Values Conversion Rule on page 30 for more information about converting data with null and empty values.

- Special symbols control.

  See Special Symbols Conversion Rule on page 32 for more information about converting special symbols.

Supported Schema Types for Formatting XML Data

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports mapping XML data with the following schema types:

- Generic
- XSD
- Java Classes

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports different implementation mechanisms when converting between JSON data and XML data based on the XSD schema type. You can select an implementation mechanism by configuring the com.tibco.plugin.restjson.json.111CompatibleMode property.

If you set this property to false or do not add the property, XSD schemas are used for converting data between XML and JSON.

If you set this property to true, XSD schemas are not used for converting data between XML and JSON, they are used for mapping and validating XML data. Besides, the XSD schema type decides which data types and elements are preserved according to the element definition in XSD when rendering JSON.

See Implementation Mechanism Exchange for Parsing and Rendering on page 116 for details about how to change the approach.

Generic

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports conversion of JSON strings to XML strings and vice versa.

XSD

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports converting a JSON string to the structured XML data with an XML schema and vice versa.

If you do not select the **Badgerfish** check box, the XML schema handles the JSON data based on the definition in XSD when parsing JSON data to XML data.

Java Classes

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports conversion of a JSON string to structured XML data with a POJO JAR file, and vice versa.

The plug-in generates XSD based on the given root of the Java class. The simple name of the Java class root name is mapped to the XSD root element in lowercase. The fields of the root class are mapped to the XSD sub-elements. The elements in the converted XSD are surrounded with the xsd:all group type.

Conversion is not supported for Java collection. In the mapping, the supported Java collection is surrounded with the xsd:all group type. If the **Validate Output** check box or the **Validate Input** check box is selected when running the Parse JSON activity or the Render JSON activity, a validation error message is generated.

Supported Data Types

Table 2 lists the supported data types defined in POJO for conversion.

Table 2 Data Types in POJO

Supported Java Type	XML Example	Corresponding XSD Type
Primitive Types		
boolean java.lang.Boolean	<boolean>true</boolean>	xsd:boolean
byte java.lang.Byte	<byte>22</byte>	xsd:byte
byte[]	<byte-array>AHiEFiEABQ==</byte-array>	xsd:base64Binary
double java.lang.Double	<double>456774543443.4553435</double>	xsd:double
float java.lang.Float	<float>4563443.435</float>	xsd:float
int java.lang.Integer	<int>12345678</int>	xsd:int
long java.lang.Long	<long>2344556678888786</long>	xsd:long
short java.lang.Short	<short>1445</short>	xsd:short

Table 2 Data Types in POJO (Cont'd)

Supported Java Type	XML Example	Corresponding XSD Type
java.math.BigDecimal	<big-decimal>342346.445332</big-decimal>	xsd:decimal
java.math.BigInteger	<big-int>23434224556</big-int>	xsd:integer
java.lang.String	<string>hello world</string>	xsd:string
Collection Types		
java.util.ArrayList java.util.LinkedList java.util.HashSet java.util.Vector java.util.LinkedHashSet	<linked-list> <string>apple</string> <string>banana</string> <string>orange</string> </linked-list>	unbounded element
Datetime Types		
java.util.Date	<date>2004-02-22 15:16:04.0 EST</date>	xsd:dateTime
java.util.Calendar	<date>2004-02-22 15:16:04.0 EST</date>	xsd:dateTime
Other Types		
java.net.URL	<url>http://helloworld.org/blah</url>	xsd:string
java.util.UUID	<uuid>ca05f023-e07f-4956-a6ef-14ddd23df47b</uuid>	xsd:string
javax.xml.datatype.Duration	<duration>PT1H2M</duration>	xsd:duration

Badgerfish Conversion Rule

Badgerfish conversion is a comprehensive conversion for parsing and rendering data between JSON format and XML format. The mapping covers XML namespace, XML attribute, CDATA, and so on. But XML comments and XML prologs including XML declaration and DTD are ignored during the conversion.

To apply Badgerfish conversion rules, select the **Badgerfish** check box in the **Configuration** tab in the Render JSON activity or the Parse JSON activity.

If the **Badgerfish** check box is selected, ensure that the input data follows Badgerfish conversion rules.

Table 3 lists detailed Badgerfish conversion rules.

Table 3 Badgerfish Conversion Rules (Sheet 1 of 3)

XML Node	JSON Key Style	Description	Example
Namespace	*@namespaceKeyName*	The XML namespace is converted as the JSON element key whose name begins with the at sign (@).	An XML namespace in JSON: "@xmlns:xsi":"http://www.w3.org/2001/XMLSchema-instance"

Attribute Key	*@AttributeKeyName*	The XML attribute is converted as the JSON element key whose name begins with the at sign (@).	Original XML:
			<doc id="1">
			</doc>
			Converted in JSON:
			{
			"doc":{
			"@id":"1"
			}
			}

Table 3 Badgerfish Conversion Rules (Sheet 2 of 3)

XML Node	JSON Key Style	Description	Example

Text Content	$	Use the single dollar sign ($) as the JSON element key, and the corresponding value is the text content.	Original XML:
			<doc id="1">
			A unix timestamp or any date accepted by strtotime
			</doc>
			Converted in JSON:
			{
			"doc": {
			"@id":"1",
			"$":"A unix timestamp or any date accepted by strtotime"
			}
			}

Element Tag	*tagName*	No special symbol is added for XML element name.	Original XML:
			<method id="1">
			<doc id="2">
			A unix timestamp or any date accepted by strtotime
			</doc>
			</method>
			Converted in JSON:
			{
			"method":{
			"@id":"1",
			"doc":{
			"@id":"2",
			"$":"A unix timestamp or any date accepted by strtotime"
			}
			}
			}

Table 3 Badgerfish Conversion Rules (Sheet 3 of 3)

XML Node	JSON Key Style	Description	Example

Parallel Same Element Tag	*tagName#order*	When multiple elements with the same name and at the same level in XML, use the JSON array to group the elements that have the same name.
		Note: When you set the com.tibco.plugin.restjson.json.111CompatibleMode property to true, if the Generic schema type is selected, #order is added after the tag name where the order variable guarantees that the JSON key is unique under the top level of a JSON object.
			Original XML:
			<method id="1">
			<doc id="2">
			A unix timestamp
			</doc>
			<doc id="3">
			any date accepted
			</doc>
			</method>
			Converted to JSON:
			{
			"method":{
			"@id":"1",
			"doc":[
			{
			"@id":"2",
			"$":"A unix timestamp"
			},
			{
			"@id":"3",
			"$":"any date accepted"
			}
			]
			}
			}

CDATA Text Content[1]	$$	Use the double dollar sign ($$) as the JSON element key with the corresponding value being the CDATA text content.
		Note: Only the Generic schema type supports CDATA conversion.
			Original XML:
			<doc id="1">
			<![CDATA[A unix timestamp or any date accepted by strtotime]]>
			</doc>
			Converted in JSON:
			{
			"doc":{
			"@id":"1",
			"$$":"A unix timestamp or any date accepted by strtotime"
			}
			}

1. If you convert JSON data to XML data and then convert back to JSON data with the XSD schema type, TIBCO ActiveMatrix BusinessWorks treats CDATA as a common text node.

Normal Conversion Rule

Compared with Badgerfish conversion, normal conversion is lossy but it generates much leaner and cleaner data.

If using the normal conversion rule to parse and render data:

- When repeatedly converting data from XML to JSON and then converting back to XML, the resulting XML data might be not the same as the original XML data. It depends on your input data.
- When repeatedly converting data from JSON to XML and then converting back to JSON, the resulting JSON data might not be the same as the original JSON data. It depends on your input data.
- When the Java Classes schema type is selected, normal conversion is used by default for parsing and rendering data.

The following sections outline normal conversion rules:

- Mapping from JSON Data to XML Data
- Mapping from XML Data to JSON Data

Mapping from JSON Data to XML Data

If the **Badgerfish** check box is cleared in the **Configuration** tab in the Parse JSON activity, normal conversion rules are applied during the mapping. Table 4 lists detailed normal conversion rules.

Table 4 Normal Conversion Rules for Parsing JSON

Schema Type	JSON Data	XML Data
Generic, XSD, and Java Classes	JSON name/value pair	XML element
	JSON array	XML repeating elements
	JSON name is xmlns or xmlns:*prefix*	XML namespace

1. If the StAXON check box under XSD schema type is selected, it handles JSON input with key _content_ in the normal way as a JSON key.

Mapping from XML Data to JSON Data

If the **Badgerfish** check box is cleared in the **Configuration** tab in the Render JSON activity, normal conversion rules are applied. Table 5 lists detailed normal conversion rules.

Table 5 Normal Conversion Rules for Rendering JSON

Schema Type	XML Data	JSON Data
Generic and XSD	XML element[1]	JSON name/value pair
	XML attribute	JSON name/value pair
	XML namespace	JSON name/value pair
	XML repeating elements	JSON array
	XML data types	JSON string data type
	XML mixed text node[2]	JSON array
Java Classes	XML element	JSON name/value pair
	XML repeating elements	JSON array
	XML data types	The same data type defined in the Java class

1. If the original XML element is a simple element that contains an attribute or a namespace, a _content_ key is added before the JSON content after the XML data is rendered to JSON data with the normal conversion rule.

   For example, the original XML data:

```
<root>
 <a attr="test">testA</a>
 <b xmlns="namespaces">testA</b>
</root>
```

   the converted JSON data:

```
{
  "root": {
    "a": {
      "attr": "test",
      "_content_": "testA"
    },
    "b": {
      "xmlns": "namespaces",
      "_content_": "testA"
    }
  }
}
```

2. For example, the original XML data:

```
<a>
 text1
 <b>text2</b>
 text3
</a>
```

   the converted JSON data:

```
{
 "a":{
  "b":"text2",
  "_content_":[
   "text1",
   "test3"
  ]
 }
}
```

Root Node Conversion Rule

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON provides the root node control. The following sections outline root node conversion rules:

- Root Node Mapping from JSON to XML
- Root Node Mapping from XML to JSON

Root Node Mapping from JSON to XML

The root node rules for converting data from JSON to XML are shown in Table 6.

Table 6 Root Node Conversion Rules from JSON to XML

Schema Type	Root Node Conversion Rule
Generic	When converting JSON strings to XML strings: • If the input JSON string has a single root, the root is preserved in the converted XML document. • If the input JSON data does not have a single root, the default root element, JSON, is added in the output XML document.
XSD	When converting a JSON string to structured XML data with the XSD schema type: • If the input JSON string has a single root, the root is preserved in the converted XML data. **Note:** If JSON root does not match with output editor root and either Badgerfish is not selected or StAXON is selected, the root element of schema specified in **Output** tab in the Parse JSON activity is used as wrapper in the generated XML. • If the input JSON string does not have a single root, the root element of the schema, which you specified in the **Output** tab in the Parse JSON activity, is used as the root element.

Table 6 Root Node Conversion Rules from JSON to XML (Cont'd)

Schema Type	Root Node Conversion Rule
Java Classes	When converting a JSON string to structured XML data with the Java Classes schema type: • If the input JSON string has a single root, the root is preserved in the converted XML data. **Note:** If the root of the input JSON string does not match the root of the schema you specified in the **Output** tab, errors occur when running the Parse JSON activity. • If the input JSON string does not have a single root, the root element of the schema, which you specified in the **Output** tab in the Parse JSON activity, is used as the root element.

Root Node Mapping from XML to JSON

The XML root node is preserved in the converted JSON string. If the **Remove Root** check box in the **Configuration** tab in the Render JSON activity is selected, the root is removed in the converted JSON string.

Null and Empty Values Conversion Rule

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON offers the capability to convert elements whose values are null or empty between XML and JSON. Both Badgerfish conversions and normal conversions are in line with the following rules:

- Mapping from XML Data to JSON Data
- Mapping from JSON Data to XML Data

Mapping from XML Data to JSON Data

The null and empty values conversion rules for mapping data from XML to JSON is shown in Table 7.

Table 7 Null and Empty Values Conversion Rules for Rendering JSON

Description	XML Data	JSON Data
An empty XML element[1]	<a/>	{"a":null}
An XML element without text node	<a>	{"a":""}
An XML element with whitespace text node[2]	<a> 	{"a":" "} or {"a":{"$":" "}}

1. If the Java Classes schema type is used, the converted JSON data must be {"a":""}.
2. When converting an XML element with whitespace text node to JSON data, as for the converting result, the Badgerfish conversion rule is different from the normal conversion rule:

   With the normal conversion rule selected, the converting result is {"a":" "}.

   With the Badgerfish conversion rule selected, the converting result is {"a":{"$":" "}}.

Mapping from JSON Data to XML Data

The null and empty values conversion rules for mapping data from JSON to XML is shown in Table 8.

Table 8 Null and Empty Values Conversion Rules for Parsing JSON

Description	JSON Data	XML Data
A JSON element with a null value[1]	{"a":null}	<a/>
A JSON string whose content is "null"	{"a":"null"}	<a>null

Table 8 Null and Empty Values Conversion Rules for Parsing JSON (Cont'd)

Description	JSON Data	XML Data
An empty JSON array	{"a":[]}	<a>
An empty JSON object	{"a":{}}	<a>
An empty JSON string	{"a":""}	<a>
A JSON value contains whitespace	{"a":" "}	<a>

1. If the Java Classes schema type is used, the converted XML data should be <a>null<a/>.

- If the XSD schema type or the Java Classes schema type is used when parsing JSON, no matter whether the original data is an empty JSON array, an empty JSON object, or an empty JSON string, the converting result is always <a/>.
- If the XSD schema type with StAXON is used when parsing an empty JSON array, then respective element is not generated in the output XML.

Special Symbols Conversion Rule

A JSON key might contain whitespaces and some special symbols , such as - **%,** $, #, which are not allowed in an element key or an attribute key in XML. By default, these special symbols are removed when parsing JSON. However, in StAXON implementation, exception is raised when parsing these special symbols in the JSON keys.

- If the number sign (#) occurs in the middle of a JSON key when parsing JSON with the Badgerfish conversion rule, the content behind #, including the #, is removed.

For example, the original JSON data:

```
{
 "product":{
  "name#A":"Kindle"
 }
},
```

the converted XML data:

```
<product>
      <name>Kindle</name>
</product>.
```

- If StAXON option is enabled, exception is raised when handling special symbols such as - #, %, $ ($ with text) in the JSON keys. However, for $ (single) as the JSON element key, it is handled as per Badgerfish conversion rule.

Chapter 4 Plug-in Tools

This chapter explains how to use the JSON tools and Swagger tools provided with TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON.

Topics

Using the JSON Tools

With TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON, you can generate an XML schema file from a given JSON file by using the JSON tools. JSON tools use normal conversion rules. See Normal Conversion Rule on page 25 for more information about the normal conversion rules.

Converting JSON Data to an XML Schema File

Perform the following steps to generate an XML schema file:

1. Start TIBCO Designer and create or open a project.
2. Select **Tools > JSON Tools> Generate XML Schema from JSON** from the TIBCO Designer menu. The **Generate XML Schema from JSON** dialog is displayed.
3. Specify a JSON file that you want to convert to an XML schema file. Click **Browse** next to the **JSON File** field and then navigate to the directory where the JSON file is saved. The JSON file can be any text type.

   Note: The file name cannot contain any of the following characters: /\:#
4. Specify a JSON file encoding type in the **JSON File Encoding** field. The default value is UTF-8.
5. Specify a name for the output XML schema file in the **Schema File Name** field.
6. Click **OK** to generate the XML schema file.

After the conversion is completed successfully, a dialog box is displayed to inform you that the XML schema file is generated. The output XML schema file is saved in the new automatically created JSON_GEN_Schemas folder in the project directory.

In the generated XSD:

- If a JSON object contains an array, the elements in this JSON object are surrounded with xs:sequence XSD group type. Otherwise, the JSON object is surrounded with the xs:all type.
- If the input JSON data does not have a root, the default root element, JSON, is added in the output XML schema.

Using the Swagger Tools

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports Swagger on both client and server sides. On the server side, you can export RESTful services to Swagger web server. See Chapter 7, Managing RESTful Web Services, page 77 for details. On the client side, you can use the Swagger tool to download Swagger API description file and call your RESTful web services in TIBCO ActiveMatrix BusinessWorks processes.

This section specifies how to download Swagger API description files to your TIBCO ActiveMatrix BusinessWorks process.

To download a Swagger API description file, perform the following steps:

1. Start TIBCO Designer.
2. Create or open a project.
3. Click **Tools > Swagger Tools > Load Swagger JSON files** from the TIBCO Designer menu.
4. Enter the Swagger doc URL in the **Load Swagger API description JSON file** dialog.
5. Click **OK** to download the Swagger API description file.
6. Select **Project > Save** from the TIBCO Designer menu to save the project.

When the download is completed, a Swagger JSON File folder is displayed in the Project panel, which contains the API description file. The generated file is .txt file.

When you download Swagger API description file from the Swagger doc URL, all the files are added in the Swagger JSON File folder, and the newly downloaded file overwrites the existing file with the same name.

Chapter 5 REST and JSON Palette

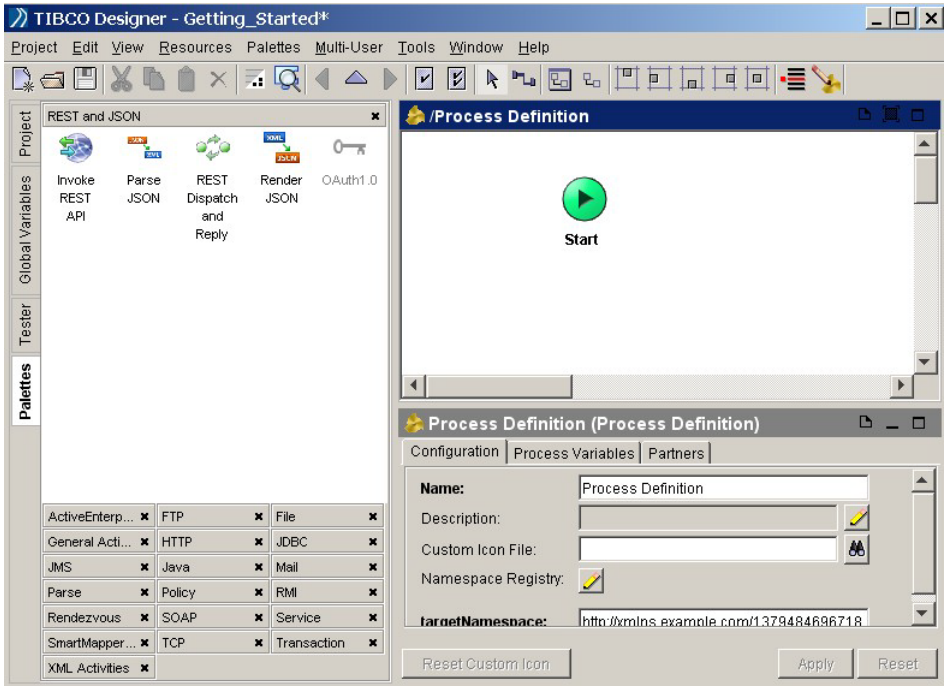
This chapter describes the REST and JSON palette, which contains one shared resource and four activities for TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON.

Topics

REST and JSON Palette Overview

The REST and JSON palette consists of one shared resource and four activities as shown in Figure 4.

Figure 4 REST and JSON Palette



Shared Resource

The OAuth 1.0 shared resource for TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON is used to set the related parameters for OAuth 1.0.

Activities

To perform different functionalities, REST and JSON activities can be divided into two groups: REST activities and JSON activities.

REST Activities

The following activities can be used to invoke RESTful web services and to expose TIBCO ActiveMatrix BusinessWorks processes as RESTful web services:

- Invoke REST API
- REST Dispatch and Reply

JSON Activities

The activities in this group are used to convert data between JSON format and XML format. The included activities in this group are as follows:

- Parse JSON
- Render JSON

OAuth1.0

Shared Resource

The OAuth1.0 shared resource allows you to set up the related parameters for OAuth 1.0.

This shared resource is used when you want to use the OAuth 1.0 authentication with the Invoke REST API activity.

Configuration

The **Configuration** tab contains the following fields as shown in Table 9. Apart from the Name an

Table 9 OAuth1.0 Configuration Tab

Field	Global Var?	Description
Name	No	The name to display as the label for the resource in the process definition.
Description	No	Short description of the resource.
You can obtain the following field information from a Service Provider.		
Consumer Key	Yes	The Consumer Key for authenticating the Consumer access to protected resources from a web service.
Consumer Secret	Yes	The Consumer Secret for authenticating the Consumer access to protected resources from a web service.
Access Token	Yes	The Access Token for authenticating the Consumer access to protected resources from a web service.
Token Secret	Yes	The Token Secret for authenticating the Consumer access to protected resources from a web service.
Signature Methods	No	A signature method used to sign the request.

Invoke REST API

Activity

The Invoke REST API activity calls RESTful web services and receives responses from the service provider.

Configuration

The fields on the **Configuration** tab are described in Table 10.

Table 10 Invoke REST API Configuration Tab (Sheet 1 of 4)

Field	Global Var?	Description
Name	No	The name to display as the label for the activity in the process definition.
Description	No	Short description of the activity.
Protocol	No	TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports the following types of protocols: • None: Indicates a RESTful web service without WADL or Swagger files. • WADL: Using the standard WADL files to describe RESTful web service. • Swagger: Using Swagger to describe RESTful web service.
Resource URI	Yes	This field specifies the root and path of a URI resource. For example, http://api.linkedin.com/v1/people/~. **Note:** This field is available only when the protocol type is None.

Table 10 Invoke REST API Configuration Tab (Sheet 2 of 4)

Field	Global Var?	Description
Method	Yes	The following REST methods are used for the requests: • GET • POST • PUT • DELETE • PATCH **Note**: If the protocol type is WADL or Swagger, only the methods that correspond to the API are listed in the **Method** field.
Response Type	Yes	The response data type, which the Service Provider supports. TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON provides the following options: • Default: To use the default data type specified by the Service Provider to return data • XML: the Service Provider returns data in XML format • JSON: the Service Provider returns data in JSON format • Binary: the Service Provider returns data in Binary format
Rich Output	No	Selecting this check box enables you to customize the data structure of the output activity in the **Output Editor** tab. See Output Editor on page 54 for more information. If the response data is in binary format, ensure that the **Rich Output** check box is selected.
Trust HTTPS Host	Yes	Selecting this check box enables you skip to verify the host name of the Service Provider.

Table 10 Invoke REST API Configuration Tab (Sheet 3 of 4)

Field	Global Var?	Description
Authentication	No	The authentication types that are supported by the Service Provider. The following are the authentication types supported by TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON: • No Authentication • Basic Authentication • OAuth 1.0 • OAuth 2.0
WADL File	No	The path and name of a WADL file. The Invoke REST API activity refines the process input from the WADL file. **Note**: This field only appears when you select WADL as the protocol type.
Swagger File	No	The path and name of a Swagger file. The Invoke REST API activity refines the process input from the Swagger file. You can reuse the Swagger API description file that was downloaded using Swagger Tool. Rename the .txt file to .json to configure Invoke REST API activity. **Note**: This field only appears when you select Swagger as the protocol type. This activity supports Swagger 1.2 and 2.0 specifications.
Base URL	No	The base URL to define the RESTful web service that you want to invoke. Selects a base URL from the list.
Paths	No	The resource path of a RESTful web service. This field is automatically filtered if a WADL or Swagger file is given.
Method ID	No	The Method ID to define the API that you want to invoke. Selects a method ID from the list.
Fields displayed when Basic Authentication is selected as the authentication type. You can obtain the field information from a Service Provider.		
Username	Yes	A valid user name for accessing a website.
Password	Yes	A valid password for accessing a website.

Table 10 Invoke REST API Configuration Tab (Sheet 4 of 4)

Field	Global Var?	Description
Field displayed when OAuth 1.0 is selected as the authentication type.		
OAuth 1.0 Parameters	Yes	The OAuth1.0 shared resource, which specifies OAuth 1.0 related parameters. See OAuth1.0 on page 40 for more information about the OAuth1.0 shared resource.
Fields displayed when OAuth 2.0 is selected as the authentication type. You can obtain the field information from a Service Provider.		
Access Token Position	Yes	Specifies one of the following ways to send the access token along with your HTTP request: **Header** Send the access token in an HTTP header. **Query** Include the access token as a URL query parameter.
Access Token Name	Yes	The name authorized by the authorization server's token endpoint.
Access Token Value	Yes	The value authorized by the authorization server's token endpoint.

Reload Button

If the WADL or Swagger file is changed, you must use the **Reload** button to refresh the WADL or Swagger file to update the **Configuration** and **Input** tabs.

Sending Data in the HTTP request

The Invoke REST API activity contains different types of parameters for sending data in a request:

- Template - This parameter is displayed when using a WADL or Swagger file to invoke RESTful web service.
- Query
- Header

- Body - This parameter contains the following child parameters:
  - Form
  - Text
  - Binary
  - Multipart

Describing RESTful Web Service with WADL or Swagger Files

If the protocol type is WADL or Swagger, and a WADL file or Swagger file is given, the WADL file or Swagger file is mapped to the input parameters in the **Input** tab.

You cannot alter the structure of the HTTP request message in the **Input** tab. If the data structure in the WADL file or Swagger file does not match the RESTful web service, edit the elements in the file and click **Reload**.

Table 11 shows the rules for mapping a WADL file to the input parameters.

Table 11 Rules for Mapping a WADL File to the Input Parameters (Sheet 1 of 3)

Input Parameter	Mapping Rule	Example
Template	Specifies one or more variables in the resource path. In a WADL file, if the style attribute value is template, the name of the parameter is mapped to the Template parameter in the **Input** tab. **Note**: During runtime, the parameters can be replaced with the input values in the **Input** tab.	In a WADL file: <resource path="/resource2/{format}"> <param **name="format"** required="true" type="xsd:string" **style="template"** default="json"> In this example, the format parameter is mapped to the input parameter.

Table 11 Rules for Mapping a WADL File to the Input Parameters (Sheet 2 of 3)

Input Parameter	Mapping Rule	Example
Query	In a WADL file, if the style attribute value is query, two parts in a WADL file are mapped to the Query parameter: • The param node directly under the resource tag: In this case, the Query parameter is shared by all the method nodes under this resource tag. • The param node under the request tag: In this case, the Query parameter is only applicable for this method.	The following examples are given based on the two mapped parts of a WADL file: • The following example shows the param node directly under the resource tag in a WADL file: <resource path="/resource2/{format}"> <param name="param1" required="true" type="xsd:string" **style="query"** default="12345"> • The following example shows the param node under the request tag in a WADL file: <method> <request> <param name="param1" required="true" type="xsd:string" **style="query"** default="12345">
Header	In a WADL file, if the style attribute value is header, the name of the parameter is mapped to the Header parameter.	In a WADL file: <resource path="/resource2/{format}"> <param name="header1" required="true" type="xsd:string" **style="header"** default="12345">
Body	The XML nodes under the representation tag are mapped to the Body parameters.	
The following parameters are child parameters of the Body parameter.		
Form	In a WADL file, if the style attribute value is Query and the nodes are under the representation tag, the name of the parameter is mapped to the Form parameter, and is encoded in application/x-www-form-urlencoded.	In a WADL file: <request> **<representation>** <param name="param2" required="true" type="xsd:string" style="query" default="12345">
Text	This input parameter always shows in the **Input** tab.	

Table 11 Rules for Mapping a WADL File to the Input Parameters (Sheet 3 of 3)

Input Parameter	Mapping Rule	Example
Binary	This input parameter always shows in the **Input** tab.	
Multipart	This input parameter always shows in the **Input** tab.	

Table 12 shows the rules for mapping a Swagger file to the input parameters.

Table 12 Rules for Mapping a Swagger File to the Input Parameters

Input Parameter	Mapping Rule	Example
Template	In a Swagger file v1.2, if the paramType parameter value is path, the name of the parameter is mapped to the Template parameter in the **Input** tab. In a Swagger file v2.0, the paramType parameter is renamed to in.	In a Swagger file v1.2: "parameters": [{ "name": "petId", "description": "Pet id to delete", "required": true, "type": "string", "**paramType**": "**path**", "allowMultiple": false }] In a Swagger file v2.0: "parameters": [{ "name": "petId", "**in**": "**path**", "description": "Pet id to delete", "required": true, "type": "integer", "format": "int64" }] In this example, the path parameter is mapped to the Template input parameter.

Table 12 Rules for Mapping a Swagger File to the Input Parameters (Cont'd)

Input Parameter	Mapping Rule	Example
Query	In a Swagger file v1.2, if the paramType parameter value is query, the name of the parameter is mapped to the Query parameter in the **Input** tab. In a Swagger file v2.0, the paramType parameter is renamed to in.	In a Swagger file v1.2: "parameters": [{ "name": "status", "description": "Status values that need to be considered for filter", "defaultValue": "available", "required": true, "type": "string", "**paramType**": "**query**" }] In a Swagger file v2.0: "parameters": [{ "name": "username", "**in**": "**query**", "description": "The user name for login", "required": true, "type": "string" }] In this example, the query parameter is mapped to the input parameter.

Table 12 Rules for Mapping a Swagger File to the Input Parameters (Cont'd)

Input Parameter	Mapping Rule	Example
Header	In a Swagger file v1.2, if the paramType parameter value is header, the name of the parameter is mapped to the Header parameter in the **Input** tab. In a Swagger file v2.0, the paramType parameter is renamed to in.	In a Swagger file v1.2: "parameters": [{ "name": "access-token", "**paramType**": "**header**", "required": false, "type": "string", "allowMultiple": false, "description": "Access token in request header" }] In a Swagger file v2.0: "parameters": [{ "name": "api_key", "**in**": "**header**", "required": false, "type": "string" }] In this example, the header parameter is mapped to the input parameter.

Table 12 Rules for Mapping a Swagger File to the Input Parameters (Cont'd)

Input Parameter	Mapping Rule	Example
The following parameters are child parameters of the Body parameter.		
Form	In a Swagger file v1.2, if the paramType parameter value is form, the name of the parameter is mapped to the Form parameter in the **Input** tab. In a Swagger file v2.0, the paramType parameter is renamed to in.	In a Swagger file v1.2: "parameters": [{ "name": "name", "description": "Updated name of the pet", "required": false, "type": "string", "**paramType**": "**form**" }] In this example, the form parameter is mapped to the input parameter. In a Swagger file v2.0: "parameters": [{ "name": "name", "**in**": "**formData**", "description": "Updated name of the pet", "required": false, "type": "string" }] In this example, the formData parameter is mapped to the input parameter.
Text	This input parameter always shows in the **Input** tab.	
Binary	This input parameter always shows in the **Input** tab.	
Multipart	This input parameter always shows in the **Input** tab.	

Describing RESTful Web Service Without WADL or Swagger Files

The description for the input parameters are shown in Table 13 for when the Protocol type is None.

You cannot alter the structure of the HTTP request message in the **Input Editor** tab. However, you can alter the child items under the three main parameters: Query, Header and body (including Form, Text, Binary, and Multipart). For example, add another param item under the Query parameter.

Table 13 Description of the Input Parameters When WADL or Swagger Files are Unused

Input Item	Description
Query	Specifies a URI query parameter for all methods that apply to the URI resource. The Query parameter is appended to the URL and is encoded in application/x-www-form-urlencoded.
Header	Specifies an HTTP header for use in the request.
Body	Specifies the body of the HTTP request message.
The following parameters are child parameters of the Body parameter.	
Form	Specifies the name/value pairs that represent the body of the HTTP message.
Text	Specifies the HTTP request message body. **type** This field must be defined according to content type encoding required by request. For example, content : type text : text/html json : application/json **content** The content of the data for HTTP request.
Binary	Specifies the binary attachment of the HTTP message. **type** In this field, specify the file name of the HTTP message attachment. TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON converts the file name to the MIME type automatically. **content** The content of the HTTP message attachment. **Note:** If type and content parameters are not specified, the Binary parameter is ignored.

Table 13 Description of the Input Parameters When WADL or Swagger Files are Unused (Cont'd)

Input Item	Description
Multipart	Specifies the use of multipart/form-data encoding or multipart/mixed encoding for the HTTP request. The default setting is multipart/form-data encoding. See http://www.faqs.org/rfcs/rfc2388.html for more information. **Note:** If the Service Provider requires the multipart/mixed encoding, add a header x with the name Content-Type and set the value of this parameter to multipart/mixed.

Input Editor

The **Input Editor** tab describes the data structure for the HTTP request message in the **Input** tab. If the Protocol type is WADL or Swagger, you can alter the element structure under the these parameters: Query, Header, and Body. See Table 13 on page 51 for more details about each parameter.

For each element you add, you must provide a name, data type, and the parameter details such as required, optional, or repeating.

For more information about how to use the **Input Editor** tab, see "Specifying Data Schema" in *TIBCO ActiveMatrix BusinessWorks™ Palette Reference*.

Input

The **Input** tab contains the following fields as shown in Table 14.

Table 14 Invoke REST API Input

Input Item	Description
URI	The root and the path of a URI resource. The URI resource is set in the **Resource URI** field in the **Configuration** tab, but you can override the value by specifying the URI resource in this field.
Timeout	The amount of time (in milliseconds) to wait for a response from the server. The default value is **0**.
Parameters	The items displayed under the Parameters item vary depending on whether the **Enable WADL** check box in the **Configuration** tab is selected. See Sending Data in the HTTP request on page 44 for detailed information about the items.
The following items only appear in the Activity Input panel when the OAuth 1.0 authentication type is selected in the Configuration tab.	

Table 14 Invoke REST API Input (Cont'd)

Input Item	Description
ConsumerKey	The Consumer Key provided by a web service. The Consumer Key is set in the specified OAuth1.0 shared resource, but you can override the value by specifying a Consumer Key in this field.
ConsumerSecret	The Consumer Secret provided by a web service. The Consumer Secret is set in the specified OAuth1.0 shared resource, but you can override the value by specifying a Consumer Secret in this field.
AccessToken	The Access Token provided by a web service. The Access Token is set in the specified OAuth1.0 shared resource, but you can override the value by specifying an Access Token in this field.
TokenSecret	The Token Secret provided by a web service. The Token Secret is set in the specified OAuth1.0 shared resource, but you can override the value by specifying a Token Secret in this field.
The following items only appear in the Activity Input panel when the OAuth 2.0 authentication type is selected in the Configuration tab.	
OAuth2TokenName	The name of the access token provided by the authorization server's token endpoint. The access token name is set in the **Token Name** field in the **Configuration** tab, but you can override the value by specifying an access token name in this field.
OAuth2TokenValue	The value of the access token provided by the authorization server's token endpoint. The access token value is set in the **Token Value** field in the **Configuration** tab, but you can override the value by specifying an access token value in this field.
The following items only appear in the Activity Input panel when the Basic Authentication type is selected in the Configuration tab.	
username	A valid user name for accessing a website. The user name is set in the **Username** field in the **Configuration** tab, but you can override the value by specifying a user name in this field.
password	A valid password for accessing a website. The password is set in the **Password** field in the **Configuration** tab, but you can override the default by specifying a password in this field.

Output Editor

The **Output Editor** tab describes the data structure in the **Output** tab. It allows you to add headers and set the response data type.

For each element you add or edit, you must provide a name, data type, and whether the parameter is required, optional, or repeating.

This tab is available only when the **Rich Output** check box is selected in the **Configuration** tab. When the **Rich Output** option is enabled, the output varies depending on the schema type defined in the **Output Editor** tab.

For more information about how to use the **Output Editor** tab, see "Specifying Data Schema" in *TIBCO ActiveMatrix BusinessWorks™ Palette Reference.*

Output

The **Output** tab contains the following fields as shown in Table 15.

Table 15 Invoke REST API Output

Output Item	Description
Status Code	HTTP Response Codes.
msg	The response message from the Service Provider.
The following fields appear when you select the **Rich Output** check box in the **Configuration** tab .	
ReasonPhrase	The reason phrase of respond codes.
header	This node contains the information of the response header.
body	This node contains the information of the response body.

Error Output

The **Error Output** tab lists the exceptions that can be generated by the Invoke REST API activity. See Trace Messages on page 133 for more information about error codes.

Table 16 Invoke REST API Error Output

Exception	Cause
JSONRestException	An exception occurs when invoking REST API during the runtime.

Table 16 Invoke REST API Error Output

Exception	Cause
ActivityTimeoutException	An exception occurs when the response is timed out.

REST Dispatch and Reply

Activity

The REST Dispatch and Reply activity exposes TIBCO ActiveMatrix BusinessWorks processes as RESTful web services. With this activity, users can map HTTP requests to TIBCO ActiveMatrix BusinessWorks processes. Once a request is received, the REST Dispatch and Reply activity calls the bound process and takes the output of the activities in the called process as the reply.

You can also use the REST Dispatch and Reply activity to handle cross-origin requests. See Support for Cross-Origin Requests on page 131 for details about the rules that the activity uses to handle cross-origin requests.

Configuration

The **Configuration** tab contains the following fields as shown in Table 17.

Table 17 REST Dispatch and Reply Configuration Tab

Field	Global Var?	Description
Name	No	The name to display as the label for the activity in the process definition.
Description	No	A short description of this activity.
Reply For	Yes	The HTTP Receiver process starter that received the requests. This is a selection list of available HTTP Receiver activities that can receive REST Resource requests.
Enable WADL Reference	No	Selecting this check box enables the use of existing WADL files to describe a RESTful web service for binding TIBCO ActiveMatrix BusinessWorks processes. The RESTful web service is loaded in the **Service Editor** tab and the Service Editor becomes read-only. Clearing the check box enables the use of a custom RESTful web service created in the **Service Editor** tab for binding TIBCO ActiveMatrix BusinessWorks processes. For more details, see Creating a RESTful Web Service with Service Editor on page 79.

Table 17 REST Dispatch and Reply Configuration Tab (Cont'd)

Field	Global Var?	Description
WADL Reference	No	Specify a WADL file that you want to use. The WADL file specified here can either be a standard one or a custom one created by using the WADL palette. For details, see Creating a RESTful Web Service with a WADL Palette on page 80. **Note:** This field is available only when the **Enable WADL Reference** check box is selected.
Enable OAuth	No	Selecting this check box enables the use of OAuth. After enabling the OAuth, the REST Dispatch and Reply activity validates the access token sent from the request that calls the APIs during runtime.
OAuth Server URL	Yes	This field is only available after you select the **Enable OAuth** check box. Specify the OAuth server URL. The plug-in connects to the OAuth server based on the URL.

Service Editor

On the **Service Editor** tab you can bind HTTP requests to corresponding TIBCO ActiveMatrix BusinessWorks processes. See Exposing BusinessWorks Processes as RESTful Web Services for details about how to bind HTTP requests to TIBCO ActiveMatrix BusinessWorks processes to expose TIBCO ActiveMatrix BusinessWorks processes as RESTful web services.

Export WADL Button

You can export a RESTful web service to a WADL file. For more information, see Exporting RESTful Web Services to a WADL File on page 88.

To export a RESTful web service to a WADL file, click the **Export WADL** button. Then, on the **Export WADL** dialog box, provide information as per Table 18.

Table 18 Export WADL

GUI Element	Description
Export Dir	The directory to which the WADL file is exported.
File Name	The name of the exported WADL file.

Table 18 Export WADL (Cont'd)

GUI Element	Description
Keep Global Variables	Select the check box to export the names of corresponding global variables that you have defined. Clear the check box to export only the values of corresponding global variables that you have defined.

Export Swagger Button

You can export RESTful web service to a Swagger API description file. For more information, see Exporting RESTful Web Service to Swagger on page 89.

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON uses Swagger version 2.0 to export Swagger API description files.

To export RESTful web services to a Swagger web server, click the **Export Swagger** button. Then, on the **Export Docs** dialog box, provide information as per Table 19.

Table 19 Export Docs

GUI Element	Description
Export As	(Required) Specify a file format for export. The following options are available: • **ZIP**: The ZIP file contains an API description file, Swagger library, and a light-weight HTTP server named Winstone. • **WAR**: The WAR file can work with any J2EE container such as Jetty and Tomcat. You can use your own server instead of the default Winstone by using this format.
REST API Host	(Required) The host where your REST APIs reside. The default is the one in the Rest Service base URL.
Swagger Server Port	(Required) The Swagger server port. The default value is 9999. This field is available only when you select the ZIP exporting format.
Export Dir	(Required) The directory where the file is exported to.
ZIP/WAR Name	(Required) The name of the exported file. The default is api-docs.

Table 19 Export Docs (Cont'd)

GUI Element	Description
JRE Home	(Optional) Specify the location where JRE is installed. This field is available only when you select the ZIP export format.
Keep Global Variables	(Optional) Select the check box to export the names of corresponding global variables you have defined. Clear the check box to export only the values of corresponding global variables that you have defined.
Access Token Position	(Optional) Specify where the access token is located in an HTTP request: HTTP header, HTTP query, or both. This field is available only when you enable OAuth.
Show Advanced	(Optional) Select this check box if you want to customize the Swagger web page.
After selecting the **Show Advanced** check box, the following fields are displayed.	
API Version	(Optional) The current version of the API exported to Swagger.
Title	(Optional) The title of the API exported to Swagger. This title is displayed at the top of the Swagger web page.
Description	(Optional) A brief description of the API exported to Swagger. This description is displayed next to the API tile. You can define styles in the description by inserting HTML tags, such as, <a>.
Terms of Service	(Optional) The link that direct to the terms of service.
Contact Name	(Optional) The name of the API contact.
Contact Email	(Optional) The email address of the API contact.
Contact URL	(Optional) The URL of the API contact.
License Name	(Optional) The name of the license type used in the API.
License URL	(Optional) The URL of the license used in the API.

The RESTful web service can be either an existing service described by the referenced WADL file or a custom one created in the **Service Editor** tab. See Creating a RESTful Web Service with Service Editor on page 79 for details about creating a RESTful web service.

If the WADL file needs modification, you can edit the WADL file separately. See Editing a RESTful Web Service for more details.

Overview

The **Overview** tab displays the general binding information with pairs of the RESTful web service and the bound TIBCO ActiveMatrix BusinessWorks process.

The RESTful web service is displayed in the following format:

method||methodID||resource

Output

The **Output** tab contains the following fields as shown in Table 20.

Table 20 REST Dispatch and Reply Output

Output Item	Description
RESTCalledInfo	The information regarding the invoked processes.
RESTResponse	The response information of an executed REST service.

Error Output

The **Error Output** tab lists the exceptions that can be generated by the REST Dispatch and Reply activity. See Trace Messages on page 133 for details about error codes.

Table 21 REST Dispatch and Reply Error Output

Exception	Description
msg	The response message from the REST Dispatch and Reply activity.
msgCode	The error code of the message.

Parse JSON

Activity

The Parse JSON activity takes JSON data, processes it, and turns it into XML data.

Configuration

The **Configuration** tab contains the following fields as shown in Table 22.

Table 22 Parse JSON Configuration Tab

Field	Global Var?	Description
Name	No	The name to display as the label for the activity in the process definition.
Description	No	Short description of this activity.
Schema Type	No	Specifies a schema type for the output data. The supported schema types are as follows: • Generic • XSD • Java Classes The default value in this field is the XSD schema type. See Supported Schema Types for Formatting XML Data on page 18 for more information.

Table 22 Parse JSON Configuration Tab (Cont'd)

Field	Global Var?	Description
Badgerfish	Yes	Specifies the conversion rules to follow: • Select this check box to enable the conversion of JSON data to XML data with the mapping of XML namespace, XML attribute, CDATA, and so on. See Badgerfish Conversion Rule on page 21 for more information. • Clear this check box to enable normal conversion rules for mapping data from JSON to XML. By default, the **Badgerfish** check box is cleared. For more information, see Mapping from JSON Data to XML Data on page 25. This check box is only available when one of the following schema types is selected: • Generic • XSD
Validate Output	Yes	Selecting this check box enables validation output data against the output schema. By default, the **Validate Output** check box is cleared. This check box is only available when one of the following schema types is selected: • XSD • Java Classes
Jar Location	No	The absolute path and name of a POJO class resource, which is used to define a schema for the output data. The POJO class resource should be in JAR format. This field is only available when the Java Classes schema type is selected.
Root Class	No	The class that points to a root Java class. This field is only available when the Java Classes schema type is selected.
StAXON	No	This check box is available when the schema type is XSD. You can select the check box for parsing larger JSON payloads to XML more efficiently. **Note:** User can configure the com.tibco.plugin.restjson.json2xml.preferNamespaceFromSchema property to prefer namespace from schema. Refer to Preferring Namespace from Schema Using StAXON.

Input

The **Input** tab contains the following field as shown in Table 23.

Table 23 Parse JSON Input

Input Item	Description
jsonString	The input JSON data for translation.

Output Editor

The **Output Editor** tab allows you to define or reference an XML schema for the activity output. You can use the JSON Tools to generate an XML schema file with a given JSON file.

This tab is only available when the XSD schema type is selected in the **Configuration** tab.

See Using the JSON Tools on page 34 for more information about the JSON Tools. See "Specifying Data Schema" in *TIBCO ActiveMatrix BusinessWorks™ Palette Reference* for more information about the Output Editor.

Output

The output for the Parse JSON activity varies depending on the data schema you specified in the **Schema Type** field in the **Configuration** tab.

Error Output

The **Error Output** tab lists the exceptions that can be generated by the Parse JSON activity. See Trace Messages on page 133 for more information about error codes.

Table 24 Parse JSON Error Output

Exception	Cause
JSONParseException	An exception occurs when parsing JSON data.
ValidationException	An exception occurs when validating the output data.

When using a POJO JAR file to format the XML data, TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON only supports class names in lowercase in the input JSON data.

Render JSON

Activity

The Render JSON activity renders XML data as a JSON string.

Render JSON

When you use the Render JSON activity to render an XML HTTP URL to a JSON string, the forward slashes are not escaped by default. See Escaping Forward Slash on page 121 for details about how to escape a forward slash.

Configuration

The **Configuration** tab contains the following fields as shown in Table 25.

Table 25 Render JSON Configuration Tab

Field	Global Var?	Description
Name	No	The name to display as the label for the activity in the process definition.
Description	No	Short description of this activity.
Schema Type	No	Specifies the schema type for the input data. The supported schema types are as follows: • Generic • XSD • Java Classes See Supported Schema Types for Formatting XML Data on page 18 for more information.

Table 25 Render JSON Configuration Tab (Cont'd)

Field	Global Var?	Description
Badgerfish	Yes	Specifies the conversion rules to follow: • Selecting this check box enables the conversion of XML data to JSON data with the mapping of XML namespace, XML attribute, CDATA, and so on. See Badgerfish Conversion Rule on page 21 for more information. • Clearing this check box enables normal conversion rules for mapping data from XML to JSON. See Mapping from XML Data to JSON Data on page 26 for more information. This check box is only available when one of the following schema types is selected: • Generic • XSD
Remove Root	Yes	Specifies whether or not to remove the root element in the output JSON string.
Validate Input	Yes	Selecting this check box enables validation input data against the input schema. This check box is only available when one of the following schema types is selected: • XSD • Java Classes
Jar Location	No	The absolute path and name of a POJO class resource that is used to define a schema for the input data. The POJO class resource should be in JAR format. This field is only available when the Java Classes schema type is selected.
Root Class	No	The class that points to a root Java class. This field is only available when the Java Classes schema type is selected.

Input Editor

The **Input Editor** tab allows you to define or reference an XML schema for the activity input. You can use the JSON Tools to generate an XML schema file with a given JSON file.

This tab is only available when the XSD schema type is selected in the **Configuration** tab.

See Using the JSON Tools on page 34 for more information about the JSON Tools. See "Specifying Data Schema" in *TIBCO ActiveMatrix BusinessWorks™ Palette Reference* for more information about the Input Editor.

Input

The input data for the Render JSON activity is in XML format; the schema type of the XML data varies depending on the schema type you specified in the **Schema Type** field in the **Configuration** tab.

Output

The **Output** tab contains the following field as shown in Table 26.

Table 26 Render JSON Output

Output Item	Description
jsonString	The translated data in JSON string format.

Error Output

The **Error Output** tab lists the possible exceptions that can be generated by the Render JSON activity. See Trace Messages on page 133 for more information about error codes.

Table 27 Render JSON Error Output

Exception	Cause
JSONRenderException	An exception occurs when rendering the data.

Chapter 6 WADL Palette

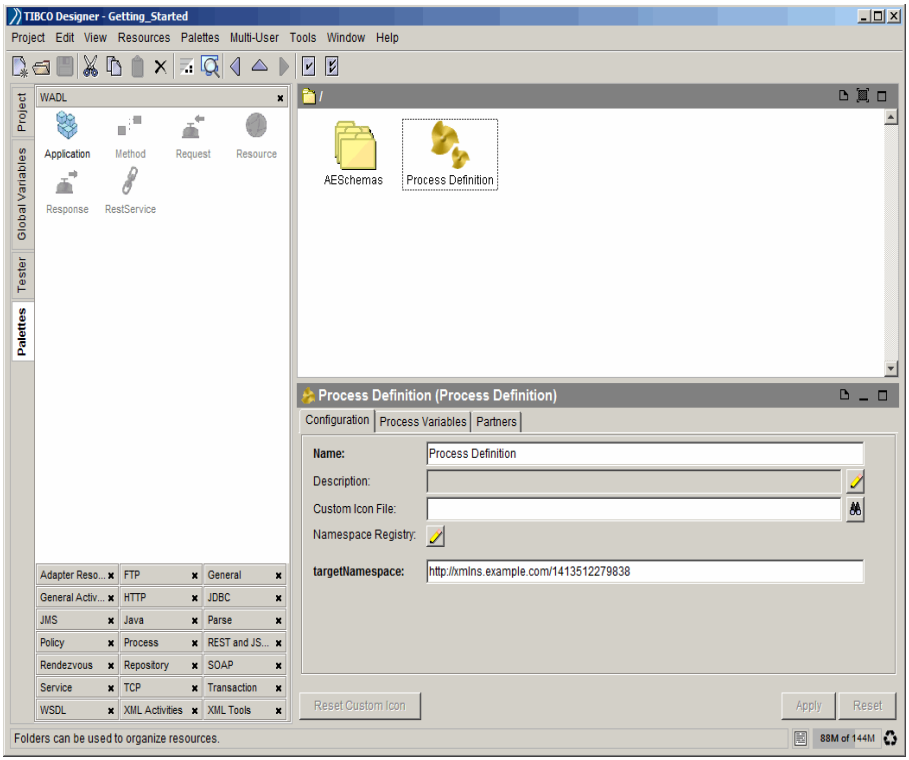
This chapter explains the WADL palette, which is used to create WADL (Web Application Description Language) files. A WADL file defines how a RESTful web service behaves and teaches clients how to interact with the service.

Topics

WADL Palette Overview

The WADL palette consists of one shared resource and a series of components used for creating a WADL file as shown in Figure 5.

Figure 5 The WADL Palette



Application

Shared Resource

A WADL file is an XML description of HTTP-based web applications, especially for RESTful web services. The Application shared resource is used to create the root element of a WADL file, which consists of a series of components.

Configuration

The **Configuration** tab contains the following fields as shown in Table 28.

Table 28 Application Configuration Tab

Field	Global Var?	Description
Application Name	No	The name of the application. The default name is Application.
Description	No	A short description. This field is optional.

RESTful Web Service

Component

A RESTful web service consists of various resources that contain specific information. The REST Service component is used to define the base URL of a RESTful web service, which is referenced by each resource added in the RESTful web service. One or more REST services can be added to an application.

See Guidelines for Creating and Editing a RESTful Web Service on page 78 for details about rules applied to a REST service element.

Configuration

The **Configuration** tab contains the following fields as shown in Table 29.

Table 29 REST Service Configuration Tab

Field	Global Var?	Description
Service Name	No	The name of the RESTful web service. The default name is RestService.
Description	No	A short description of the RESTful web service.
Base URL	Yes	(Required) The base URL of the RESTful web service.

Resource

Component

The resource element acts as a container and is the source of specific information. Each resource is referenced with a global identifier. The Resource component is used to define resource-related parameters. The resource parameters are shared by all the sub resources and methods under this resource. One or more resources can be added to a RESTful web service.

See Guidelines for Creating and Editing a RESTful Web Service on page 78 for more details about rules applied to a resource element.

Configuration

The **Configuration** tab contains the following fields as shown in Table 30.

Table 30 Resource Configuration Tab

Field	Global Var?	Description
Resource ID	No	(Required) The ID used to identify the resource.
Description	No	A short description of the resource.
Resource Path	Yes	(Required) The relative path of the resource.
Parameters	No	The parameters that are used in this resource. The parameters specified here are inherited by the methods added to this resource. Use + and ✕ next to the Parameters table to add and remove parameters; use ↑ and ↓ to arrange the order of parameters. The name and style attributes must be specified each time a parameter is added.
The following are available options of the style attribute.		
Query	No	A URI query parameter for all methods that apply to this resource.
Template	No	One or more variables defined in the resource path.
Header	No	An HTTP header for use in the request.

Method

Component

A method element is a child node of a resource. The Method component is used to define methods used for the HTTP request. One or more methods can be added to a resource.

See Guidelines for Creating and Editing a RESTful Web Service on page 78 for more details about rules applied to a method element.

Configuration

The **Configuration** tab contains the following fields as shown in Table 31.

Table 31 Method Configuration Tab

Field	Global Var	Description
Method ID	No	The ID used to identify the method.
Description	No	A short description of the method. This field is optional.
Method Name	No	The name of the method that is being used. Four methods are available: GET, POST, DELETE, and PUT.

Request

Component

The Request component is used to define the input to a method with a collection of a parameters and representations. Only one request can be added to a method.

See Guidelines for Creating and Editing a RESTful Web Service on page 78 for more details about rules applied to a request element.

Configuration

The **Configuration** tab contains the following fields as shown in Table 32.

Table 32 Configuration Tab

Field	Global Var?	Description
Request Name	No	The name of the request.
Description	No	A short description of the request. This field is optional.
Parameters	No	The parameters to be used as the input of the HTTP request. Use and next to the Parameters table to add and remove parameters; use and to arrange the order of parameters. The name and style attributes must be specified each time a parameter is added.
Representations	No	The representation of the HTTP request. By default, the request data can be any data format. Two options are available: XML and JSON. **XML** The request data is in XML format when the **XML** radio button is selected. **JSON** The request data is in JSON format when the **JSON** radio button is selected.
The following are available options of the style attribute.		
Query	No	A URI query parameter of the request.
Header	No	An HTTP header for use in the request.

Table 32 Configuration Tab (Cont'd)

Field	Global Var?	Description
Form	No	Specifies the name/value pairs that represent the body of the HTTP request.

Response

Component

The Response component is used to define the output to a method with a collection of parameters and representations. One or more responses can be added to a method.

See Guidelines for Creating and Editing a RESTful Web Service on page 78 for more details about rules applied to a response element.

Configuration

The **Configuration** tab contains the following fields.

Table 33 Configuration Tab

Field	Global Var?	Description
Response Name	No	The name of the response.
Description	No	A short description of the response. This field is optional.
Representations	No	The representation of the response. By default, the response data can be any data format. Two options are available: XML and JSON. **XML** The response data is in XML format when the **XML** radio button is selected. **JSON** The response data is in JSON format when the **JSON** radio button is selected.

Chapter 7 Managing RESTful Web Services

This chapter explains how to create and edit RESTful web services, and describes rules that apply to WADL files. This chapter also specifies how to expose TIBCO ActiveMatrix BusinessWorks processes as RESTful web services, and how to export RESTful web services to a WADL file or a Swagger file.

Topics

Guidelines for Creating and Editing a RESTful Web Service

A WADL file is used to describe RESTful web services. This section describes some rules and restrictions applied for a WADL file, which are useful when creating and editing RESTful web services.

Adhere to the following rules when creating and editing a RESTful web service:

- **RESTful web service** The base URL of the RESTful web service must be unique.
- **Resource** A resource element must adhere to the following rules:
  - The resource ID must be unique among siblings. That is, the resource ID must be different from any other ID, even a method ID, in the WADL file.
  - The resource path must be unique among siblings. That is, the resource path must be different from any other resource path under the same RESTful web service in the WADL file.
  - The values of both the name and style attributes must be unique. The values cannot be the same as that of another resource parameter and the request parameters of the method under this resource.
- **Method** A method element must adhere to the following rules:
  - The method ID must be unique among siblings. That is, the method ID must be different from any other ID, even a resource ID, in the WADL file.
  - A method element can contain only one request node.
- **Request** A request element must adhere to the following rules:
  - The request name must be unique among siblings.
  - The values of both the name and style attributes must be unique. The values cannot be the same as that of another request parameter and the method resource parameters.
- **Response** The response name must be unique among siblings.

Creating a RESTful Web Service

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports binding HTTP requests from a custom RESTful web service to TIBCO ActiveMatrix BusinessWorks processes.

You can create a custom RESTful web service in the following ways:

- Use the **Service Editor** tab of the REST Dispatch and Reply activity to create a RESTful web service. See Creating a RESTful Web Service with Service Editor for details.

  If a RESTful web service is created in this way, the binding process can be performed directly in the **Service Editor** tab. See Service Editor on page 57 for details.

- Use the WADL palette to create a WADL file to describe a custom RESTful web service. See Creating a RESTful Web Service with a WADL Palette for details.

  If a RESTful web service is created in this way, to bind the HTTP requests, you must select the **Enable WADL Reference** check box and select the created WADL file in the **WADL Reference** field in the **Configuration** tab, and then perform the binding process in the **Service Editor** tab. See REST Dispatch and Reply on page 56 for details.

Creating a RESTful Web Service with Service Editor

To create a RESTful web service, complete the following steps:

1. In TIBCO Designer, define a process and expand the **REST and JSON** palette in the Palettes panel. See Getting Started on page 5 for details about how to define a process.
2. Drag the **REST Dispatch and Reply** icon from the Palettes panel to the Design panel.
3. In the **Configuration** tab, select **HTTP Receiver** from the **Reply For** list.
4. In the **Service Editor** tab, right-click the **Application** node in the left panel, and then select **Add > RestService**. A service node with the default name, RESTService0, is added under the Application node. See RESTful Web Service on page 70 for details about the RESTful web service.

Ensure that the **Enable WADL Reference** check box is cleared in the **Configuration** tab.

5. Enter the service name and the base URL for the RESTful web service in the right panel and click **Apply** to save your configuration.

6. Right-click the newly added service node in the left panel and select **Add > Resource**. A resource node with the default name, Resource0, is added under the service node.
7. Enter a resource path and parameters to use in the right panel. See Resource on page 71 for details about the resource parameters.
8. Right-click the newly added resource node and select **Add > Method**. A method node with the default name, Method0, is added under the resource node. See Method on page 72 for details about the method element.
9. Enter a method ID and method name in the right panel.

   See Service Editor on page 57 for details about how to bind this HTTP request to a TIBCO ActiveMatrix BusinessWorks process.

The following steps: step 10 and step 11 are optional. If you want to define additional or specific parameters for a method node, except for parameters inherited from the parent resource node, you can add a request node to specify more parameters.

10. Right-click the method node, and then select **Add > Request** or **Add > Response**. A request node or a response node are added under the method node.
11. Enter parameters used for the request node or response node. See Request on page 73 and Response on page 75 for details about the request and response components.

Creating a RESTful Web Service with a WADL Palette

You can create a custom WADL file to describe a RESTful web service by using the WADL palette.

To create a WADL file, complete the following steps:

1. In TIBCO Designer, select the project folder in the Project panel. See Getting Started on page 5 for details.
2. Expand the **WADL** folder in the Palettes panel and drag the **Application** icon to the Design panel.
3. Enter a name for the web application in the Application Configuration panel and click **Apply** to save your configurations.
4. Double-click the **Application** icon in the Design panel, and then drag the **RestService** icon from the Palettes panel to the Design panel.
5. Configure the RESTful web service in the REST Service Configuration panel. See RESTful Web Service on page 70 for details about the RESTful web service.
6. Double-click the **RestService** icon in the Design panel, and then drag the **Resource** icon from the Palettes panel to the Design panel.

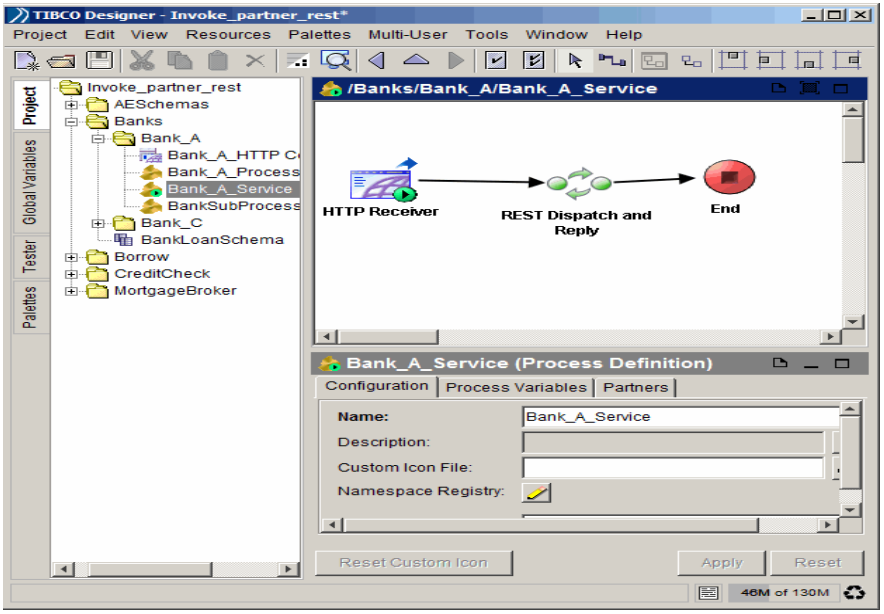
7. Configure the resource in the Resource Configuration panel. See Resource on page 71 for details about the resource parameters.
8. Double-click the **Resource** icon in the Design panel, and then drag the **Method** icon in the Palettes panel to the Design panel.
9. Configure the method in the Method Configuration panel. See Method on page 72 for details about the method element.
10. Double-click the **Method** icon in the Design panel, and then drag the **Request** icon in the Palettes panel to the Design panel.
11. Configure the request in the Request Configuration panel. See Request on page 73 for details about the request component.
12. Drag the **Response** icon from the Palettes panel to the Design panel.
13. Configure the response in the Response Configuration panel. See Response on page 75 for details about the response component.

Editing a RESTful Web Service

There are two ways to edit a RESTful web service:

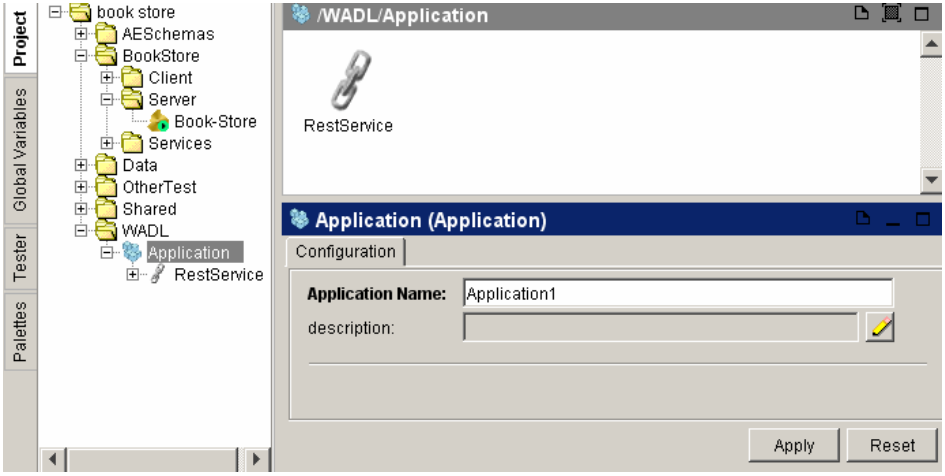
- **Use the Service Editor Tab** If the RESTful web service is created as described in Creating a RESTful Web Service with Service Editor, you can edit the RESTful web service in the **Service Editor** tab as shown in Figure 6.

Figure 6 Service Editor



- **Use the WADL Palette** If a RESTful web service is created as described in Creating a RESTful Web Service with a WADL Palette, you can edit the RESTful web service by directly clicking the corresponding element in the WADL tree as shown in Figure 7.

Figure 7 Editing a WADL File



Exposing BusinessWorks Processes as RESTful Web Services

After creating a RESTful web service, you can bind HTTP requests to TIBCO ActiveMatrix BusinessWorks processes to expose TIBCO ActiveMatrix BusinessWorks processes as RESTful web services.

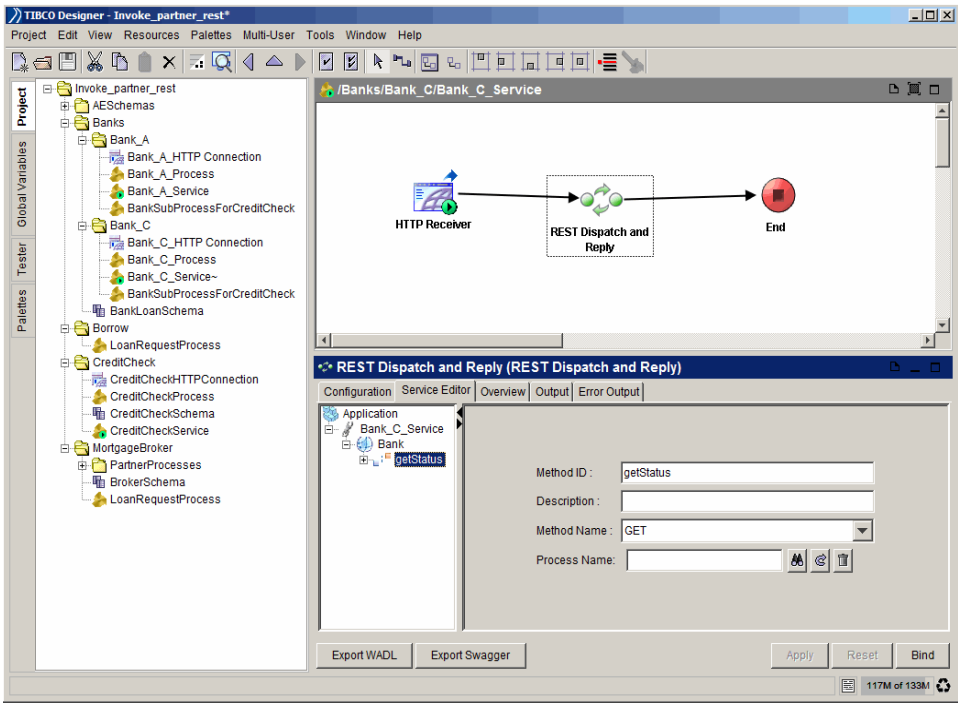
Binding HTTP Requests to BusinessWorks Processes

To bind HTTP requests to TIBCO ActiveMatrix BusinessWorks processes, complete the following steps:

1. In the left navigation pane, select the method that you want to bind and click the method node as shown in Figure 8.

   Note: One method can only bind one process.

Figure 8 Select a Method



2. In the right panel, click **Browse resources** next to the **Process Name** field. The **Select a Resource** dialog is displayed.
3. In the **Select a Resource** dialog, select the process that you want to bind to and click **OK**.

After selecting the process, you can click **Go to reference resource** next to the **Process Name** field to direct to the bound process, where you can view and change the configurations of the bound process.

If you want to remove the bound process, click **Clear reference** to clear your current selection.

4. In the right panel, click **Bind**. The **Process Binding** dialog is displayed.
5. In the **Process Binding** dialog, configure the input and output mappings, and then click **Bind** to save your configurations. See Configuring the Input and Output Binding for details.

Configuring the Input and Output Binding

You can use the **Process Binding** dialog to specify how the HTTP request and response are mapped to the bound process.

- **Input Binding** You can use it to map parameters of the HTTP request to the input of the bound process.

  The source data can be HTTP requests, Global Variables, Process Context, or other activities in this project.

- **Output Binding** You can use it to map the output of the bound process to the HTTP response as a reply. Two tabs are available in the Output Binding panel.

  — **Binding** You can use it to define how the output of the bound process is mapped to the HTTP response.

  The source data of the output can be the output of the bound process, Global Variables and Process Context in this project.

  — **ResponseHeaders** You can use it to customize the HTTP response headers.

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports:

- Customizing response code. See Customizing Response Code on page 85 for details about how to customize response codes.
- Downloading multiple part contents in one request from a TIBCO ActiveMatrix BusinessWorks process. See Downloading MIME Multiple Parts Messages on page 87 for details about how to download multiple part messages in one request from a TIBCO ActiveMatrix BusinessWorks process.

Customizing Response Code

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports the following default respond codes:

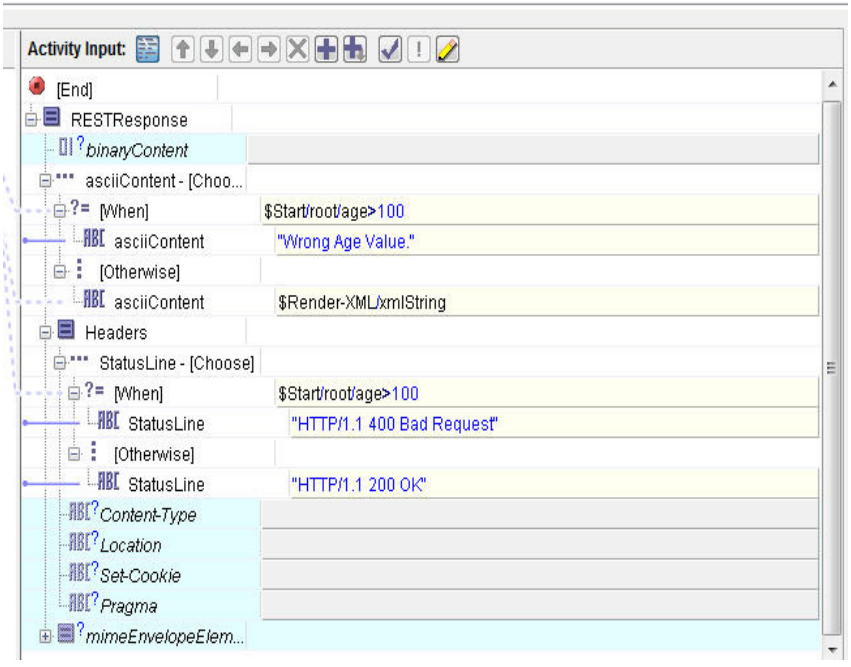
- 200 (OK)
- 400 (Bad Request)
- 404 (Not Found)
- 500 (Internal Server Error)
- 401 (Unauthorized): This respond code is available when you enable OAuth.

Additionally, the plug-in also supports customizing the 200, 400, and 500 response codes.

To customize the response codes, complete the following steps:

1. In the Output Binding panel, click the **Binding** tab.
2. In the Activity Input panel, expand **RESTResponse** > **Headers**.
3. Right-click **StatusLine**, and then select **Statement** > **Surround with choice**.
4. In the Choice dialog, specify the number of the when conditions, and whether you need the otherwise conditions, and then click **OK**.
5. Enter values in the **When**, **Otherwise** and **StatusLine** nodes accordingly as shown in Figure 9.

Figure 9 Customize the Response



Downloading MIME Multiple Parts Messages

You can download multiple part messages in one request from a TIBCO ActiveMatrix BusinessWorks process which is exposed as a RESTful Web Service.

Prerequisites

Before downloading multiple parts of messages in one request from the server side, you need to enable the handleAllMimePartsAsAttachment property in the designer.tra file:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory.
2. Open the designer.tra file, and enter the following parameters in one line:

```
bw.plugin.http.handleAllMimePartsAsAttachment=true
```

3. Save the file.

Download Multiple Parts

Configure the **mimeEnvelopeElement** item in the **Activity Input** panel of the Output Binding window to download multiple parts of the messages.

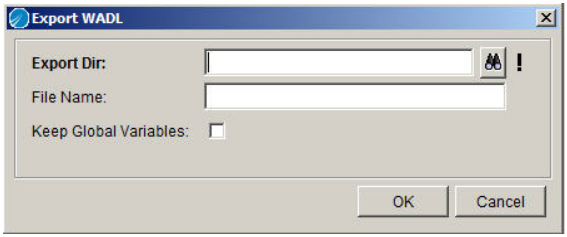
Exporting RESTful Web Services to a WADL File

TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON supports exporting a RESTful Web Services to a WADL file.

To export a RESTful Web Services to a WADL file, complete the following steps after you define a RESTful web service in the REST Dispatch and Reply activity:

1. Click the REST Dispatch and Reply activity in the process.
2. In the **Service Editor** tab, click **Export WADL**. The **Export WADL** dialog is displayed as shown in Figure 10.

Figure 10 Export WADL



3. In the Export Dir dialog:
   - Specify a directory where the WADL file can be exported.
   - Enter the name of the WADL file. Click **OK** to save the configuration.
   - Select the **Export GV** check box to export the global variables that you have defined.
4. Click **OK** to save the configuration.

Exporting RESTful Web Service to Swagger

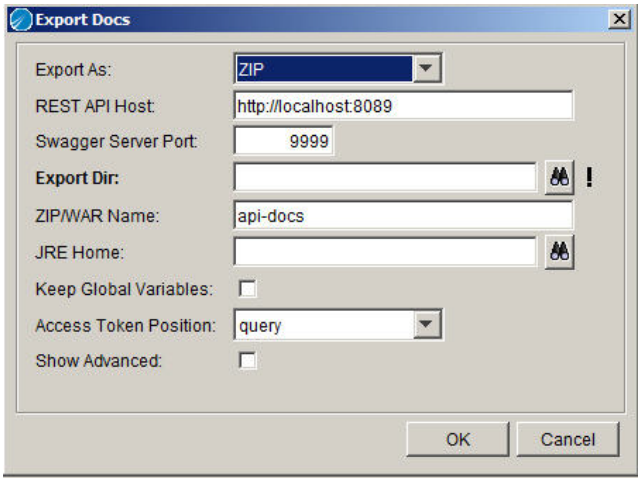
TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON supports exporting the RESTful web services to Swagger version 2.0.

In this document, Swagger refers to Swagger v2.0.

To Export the RESTful web services to Swagger, complete the following steps:

1. Create a RESTful web services in the REST Dispatch and Reply activity. See Creating a RESTful Web Service on page 79 for details about how to create a RESTful web service.
2. Click the REST Dispatch and Reply activity in the process.
3. In the **Service Editor** tab, click **Export Swagger**. The **Export Swagger** dialog is displayed as shown in Figure 11.

Figure 11 Export Swagger Dialog



4. Select an exporting format from the **Export As** list.

   See Export Swagger Button on page 58 for details about the exporting format.
5. In the **REST API Host** field, enter the REST API host where Swagger can send HTTP request to. For example, http://localhost:8089.
6. If you select the ZIP format, Enter the Swagger server port in the **Swagger Server Port** field. The default value is 9999.

Note: You can also update the server port by updating the winstone.properties file which is located in the *api-docs* directory after you export the Swagger file.

7. In the **Export Dir** field, specify a directory where the file is exported to.
8. Name the exported file in the **ZIP/WAR Name** field.
9. If you select the ZIP format, specify the JRE Home in the **JRE Home** field.
10. If you want to export the names of corresponding global variables, select the **Keep Global Variables** check box.
11. If you enable the OAuth function, specify where the access token is located in an HTTP request in the **Access Token Position** list.
12. If you want to customize the Swagger web page, select the **Show Advanced** check box and configure the corresponding fields. See Export Swagger Button on page 58 for details.
13. Click **OK** to save the configuration.

Starting a Swagger Server

After exporting the RESTful web service to Swagger, you can start a Swagger server.

The way to start a Swagger server varies depending on the exported format. If you want to use your own server instead of the default Winstone, deploy the WAR file to a J2EE container. If you use Winstone to start a Swagger server, complete the following steps:

1. Unzip the exported Swagger file.
2. Run the startup file on the command line to start the Swagger server.

Testing a RESTful Web Service in Swagger

After you start the Swagger server, complete the following steps to test RESTful web services in Swagger:

1. Open browser and enter the following URL:

   http://host:swaggerServerPort/swagger.ui

   A Swagger Web UI is displayed as shown in Figure 12.

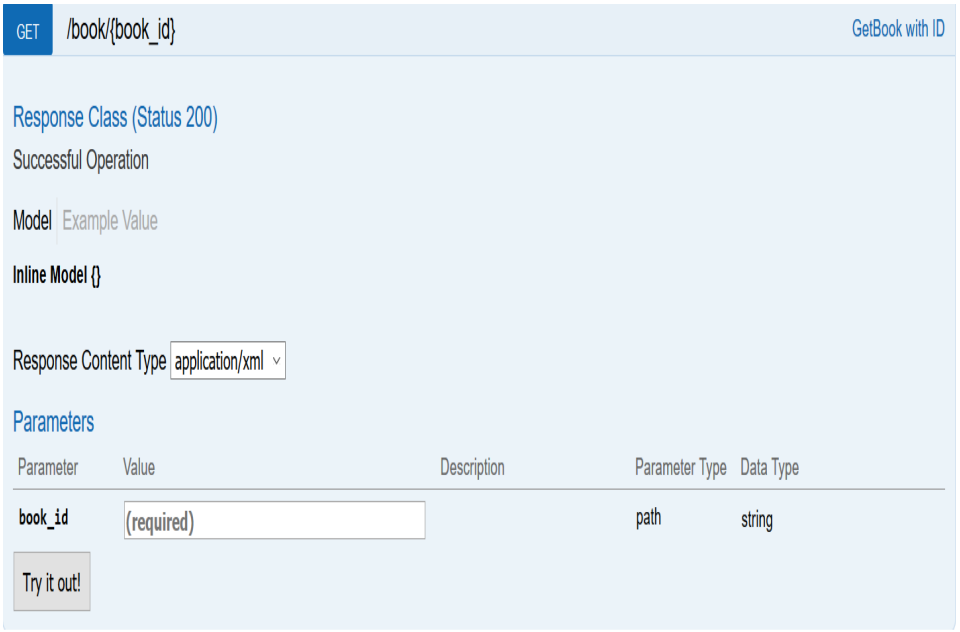
Figure 12 Swagger Web UI

You can change what is displayed on the page by clicking the following buttons:

- **Show/Hide**: Click to show or hide the APIs on the page.
- **List Operations**: Click to list all the operations of APIs.
- **Expand Operations**: Click to expand all the operations of APIs.

2. Browse the API list and expand an API. Select a response content type and parameter content type from the two lists as shown in Figure 13.

Figure 13 Expanding the API



3. Enter the required parameters, and then click **Try it out** to test the API as shown in Figure 14.

Figure 14 Entering the Parameters

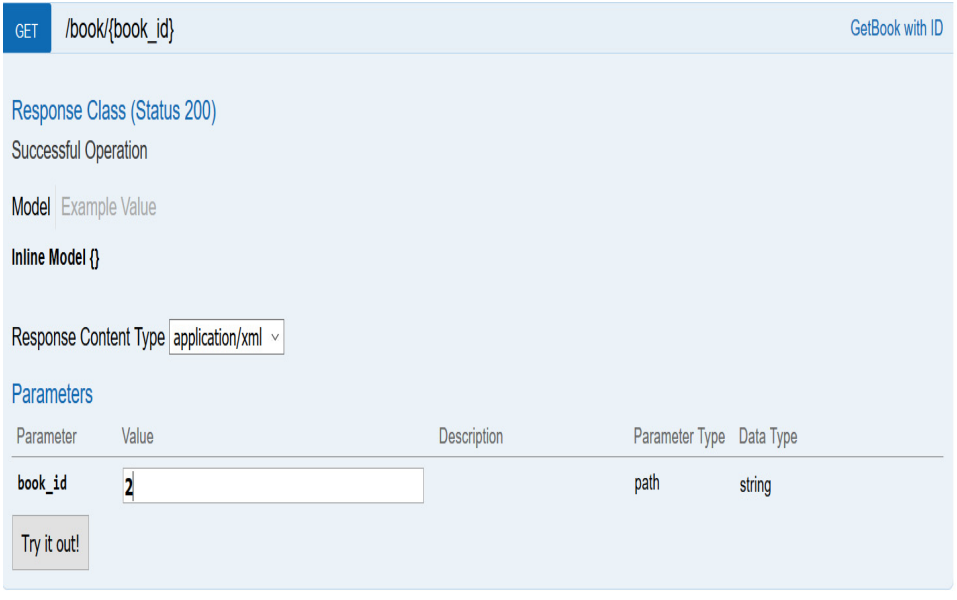
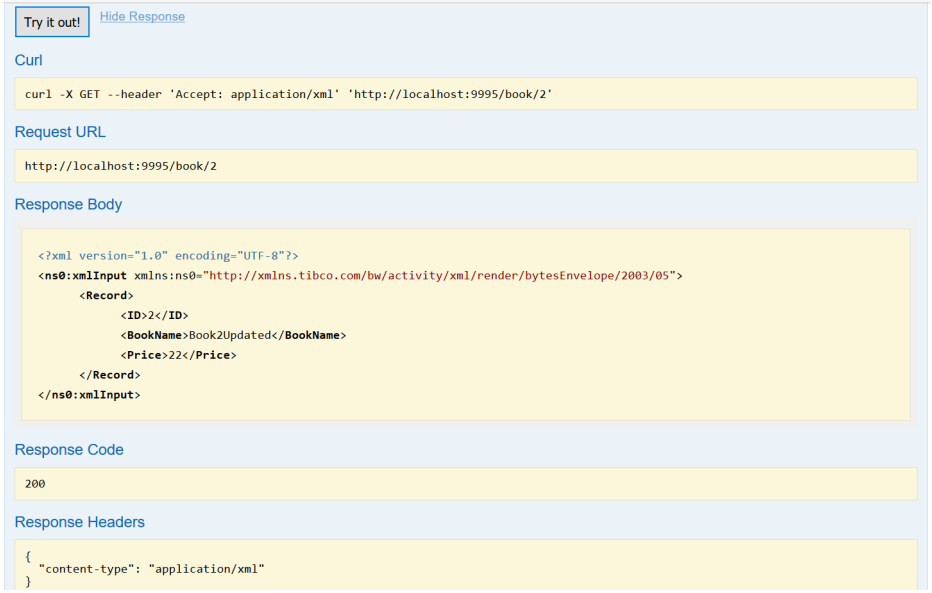


Figure 15 shows the results and whether the API is useful.

Figure 15 Checking the Results



You might fail to call the REST APIs in Swagger due to the cross-origin restrictions. See Support for Cross-Origin Requests on page 131 for details about the rules to conduct a cross-origin request.

Chapter 8 Using Sample Projects

This chapter describes sample projects packaged with TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON. Working through the sample projects helps you understand how TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON operates.

Topics

Overview of the Examples

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON examples are packaged with the installation of the plug-in. This chapter describes how to run examples on a Microsoft Windows platform.

The following examples are provided in the plug-in:

- The LinkedInPeopleSearch Project - Shows how to invoke the LinkedIn server and receives response from the server.
- The JSONSample Project - Shows how to convert data between JSON and XML with normal conversion rules and Badgerfish conversion rules.
- The InvokePartnerREST Project - Shows how to expose TIBCO ActiveMatrix BusinessWorks processes as RESTful web services and how to invoke the RESTful web service.

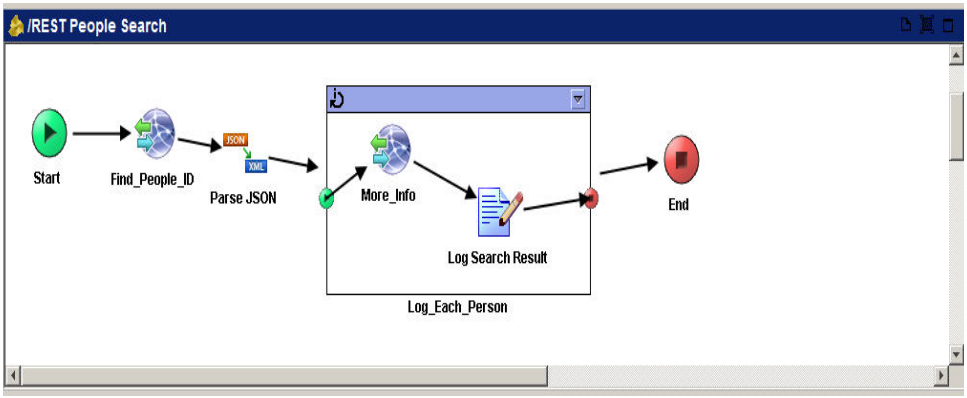
The LinkedInPeopleSearch Project

The LinkedInPeopleSearch project describes how to use TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON to invoke the LinkedIn server and receive responses from the Service Provider.

Process Description

This project contains the REST People Search process. This process invokes the LinkedIn server to find all the people whose name is John Bender. Figure 16 shows the REST People Search process definition.

Figure 16 The REST People Search Process



The process performs the following operations:

1. The Find_People_ID activity invokes the LinkedIn server with a given WADL file to find all the people whose name contains John Bender and then receives response data in JSON format from the LinkedIn server. The response data lists the name and ID of each person whose name contains John Bender.

   The given WADL file, linkedin-wadl.xml, is located in the *RESTJSON_HOME*\examples\LinkedInPeopleSearch\apigee-wadl-library\linkedin directory.

2. The Parse JSON activity converts JSON data to XML data with an XML schema file and passes the XML data to the Log_Each_Person group.

   The XML schema file, schema.xsd, is located in the *RESTJSON_HOME*\examples\LinkedInPeopleSearch\JSONSchema directory.

3. For each person listed in the converted XML data, the following activities are processed in the Log_Each_Person group:
   a. The More_Info activity invokes the LinkedIn RESTful web services with the ID of a person whose name contains John Bender, and receives the response data in XML format from the LinkedIn server and then passes the data to the Log Search Result activity. The response data lists more information about the person, John Bender . For example, the information about headline of John Bender.
   b. The Log Search Result activity logs the received data in a specified file.

Setting Up the Project

Before running the project, you need to perform the following steps to set up the project in TIBCO Designer:

1. Start TIBCO Designer.
2. Click **Open Existing Project** in the TIBCO Designer startup panel. The **Open Project** dialog is displayed.
3. Click **Browse** next to the **Project Directory** field and then select the LinkedInPeopleSearch folder, which is located in the *RESTJSON_HOME*\examples\LinkedInPeopleSearch directory. Click **OK**. The project is displayed.
4. Configure the Global Variables for a LinkedIn API key.
   a. Click the **Global Variables** tab in the Project panel.
   b. Click the **LinkedIn_OAuth** item and then click **Open Advanced Editor**  at the top of the left corner in this panel. The **Global Variables** dialog is displayed.
   c. Set values for the Access_Token, Consumer_Key, Consumer_Secret, and Token_Secret parameters under the LinkedIn_OAuth item. These values are provided by the Service Provider. In this example, you should register your application with LinkedIn to receive an API key. For more information, see https://developer.linkedin.com/documents/quick-start-guide.
   d. Click **OK** to save your setting.

5. Configure the Global variable for the generated log file of the Log Search Result activity.
   a. Click the **Global Variables** tab in the Project panel.
   b. Click the **LogFile** item and then click **Open Advanced Editor** at the top of the left corner in this panel. The **Global Variables** dialog is displayed.
   c. Expand the **LogFile** item and then set the value of the Path parameter to a directory and a file name, which you want to save the generated log file in. For example, C:\temp\Linkin.txt.
   d. Click **OK** to save your setting.
6. Select **Project > Save** from the TIBCO Designer menu to save the project.

Running the Project

After setting up the project, complete the following steps to trigger the REST People Search process:

1. In the Project panel, click the **Tester** tab and then click **Start testing viewed process** at the upper left corner in this panel. The **Select Processes To Load** dialog is displayed.
2. Select the **REST People Search (current)** check box and then click **Load Selected**. The process is started.

   When the process executes, the elements of the process change colors depending upon what is occurring in the executing process instance. If all the transition lines change to green, it means the process runs successfully.
3. Click **Stop testing** to return to the Design mode.

You can set breakpoints in the process definition at a specified point where you want to stop a running process and examine its state and process data, for example, you can set breakpoints before or after an activity.

See *TIBCO ActiveMatrix BusinessWorks™ Process Design Guide* for detailed information on using the test mode, including setting breakpoints and the element colors in the test mode.

Expected Results

After the process is successfully completed, the specified log file is generated.

The JSONSample Project

The JSONSample project shows how to use TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON to convert data between JSON and XML using normal conversion rules and Badgerfish conversion rules. In this project, the Trading Order business object is used.

Processes Description

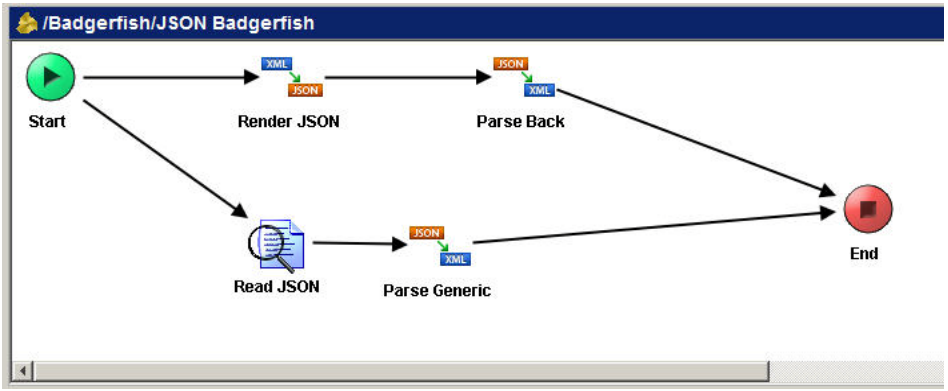
This project contains the following predefined processes:

- The JSON Badgerfish Process
- The Trading Orders Process

The JSON Badgerfish Process

The JSON Badgerfish process converts data between JSON and XML with Badgerfish conversion rules. In this process, the Generic and XSD schema types are used to construct the output data. Figure 17 shows the JSON Badgerfish process.

Figure 17 The JSON Badgerfish Process



The process contains two parallel operations:

- Converting Data from XML to JSON and then Back to XML
- Converting JSON Strings to XML Strings

Converting Data from XML to JSON and then Back to XML

In this operation, the following activities are processed:

1. The Render JSON activity takes XML data, and renders it as a JSON string. The output data is structured with an XML schema file, Book_info.xsd, which is located in the *RESTJSON_HOME*\examples\JSONSample\Badgerfish directory.
2. The Parse Back activity takes the converted JSON data and then converts it to XML data.

Converting JSON Strings to XML Strings

In this operation, the following activities are processed:

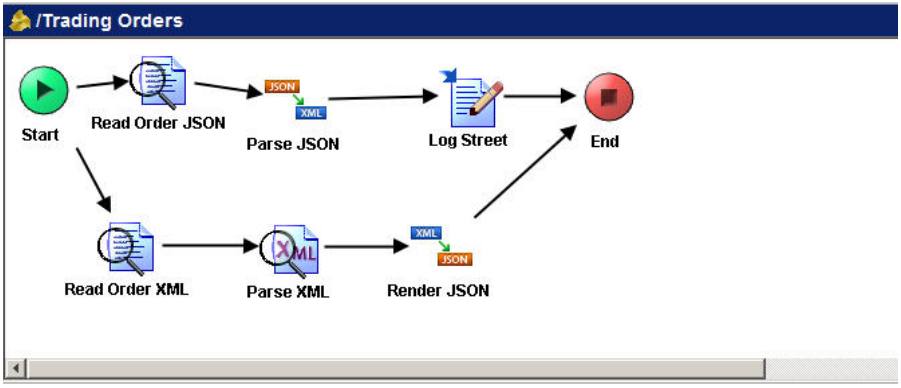
1. The Read JSON activity reads a file, Book_info.json, and passes the file content to the Parse Generic activity.

   The Book_info.json file is located in the *RESTJSON_HOME*\examples\JSONSample\Badgerfish directory.
2. The Parse Generic activity takes the JSON string from the Book_info.json file and then converts it to an XML string.

The Trading Orders Process

The Trading Orders process converts data between JSON and XML with normal conversion rules on a trading orders business object. In this process, the XSD schema type is used to format the XML data. Figure 18 shows the Trading Orders process definition.

Figure 18 The Trading Orders Process



The process contains two parallel operations:

- Converting Data from JSON to XML
- Converting Data from XML to JSON

Converting Data from JSON to XML

In this operation, the following activities are processed:

1. The Read Order JSON activity reads a JSON data file, BW_new_order.json, and passes the file content to the Parse JSON activity.

   The BW_new_order.json file is located in the *RESTJSON_HOME*\examples\JSONSample\SampleData directory.

2. The Parse JSON activity takes the JSON string from the Read Order JSON activity and then converts it to an XML string.

   The output data is structured in an XML schema file, BW_new_order.xsd, which is located in the *RESTJSON_HOME*\examples\JSONSample\JSON_GEN_Schemas directory.

3. If the Parse JSON activity executes successfully, the Log Street activity writes messages about the street.

Converting Data from XML to JSON

In this operation, the following activities are processed:

1. The Read Order XML activity reads the BW_new_order.xml data file, and passes the file content to the Parse XML activity.

   The BW_new_order.xml file is located in the *RESTJSON_HOME*\examples\JSONSample\SampleData directory.

2. The Parse XML activity takes the XML data from the Read Order XML activity and then converts it to an XML schema tree based on an XML schema file, BW_new_order.xsd, which is located in the *RESTJSON_HOME*\examples\JSONSample\JSON_GEN_Schemas directory.

3. The Render JSON activity takes the XML data from the Parse XML activity, and renders it as a JSON string.

Setting Up the Project

Before running the project, you need to perform the following steps to set up the project in TIBCO Designer:

1. Start TIBCO Designer.
2. Click **Open Existing Project** in the TIBCO Designer startup panel. The **Open Project** dialog is displayed.
3. Click **Browse** next to the **Project Directory** field and then select the JSONSample folder, which is located in the *RESTJSON_HOME*\examples\JSONSample directory. Click **OK**. The project is displayed.

4. Configure the Global Variables for a JSON file, which is used to convert to XML format.
   a. Click the **Global Variables** tab in the Project panel.
   b. Click the **JSON_Plugin** item and then click **Open Advanced Editor** at the top of the left corner in this panel. The **Global Variables** dialog is displayed.
   c. Expand the **JSON_Plugin** item and then set a value for the SampleProjectDir parameter. The value should be as follows:

      RESTJSON_HOME\examples\JSONSample\

   d. Click **OK** to save your setting.
5. Select **Project > Save** from the TIBCO Designer menu to save the project.

Running the Project

After setting up the project, complete the following steps to trigger the JSON Badgerfish process or the Trading Orders process:

1. In the Project panel, click the **Tester** tab and then click **Start testing viewed process** at the top left corner in this panel. The **Select Processes To Load** dialog is displayed.
2. Select the **Trading Orders** check box, or expand **Badgerfish** and select the **JSON Badgerfish** check box. Click **Load Selected**. The process is started.

   When the process executes, the elements of the process change colors depending upon what is occurring in the executing process instance. If all the transition lines change to green, it means the process runs successfully.
3. Click **Stop testing** to return to the Design mode.

You can set breakpoints in the process definition at a specified point where you want to stop a running process and examine its state and process data, for example, you can set breakpoints before or after an activity.

See *TIBCO ActiveMatrix BusinessWorks™ Process Design Guide* for detailed information on using the test mode including setting breakpoints and the element colors in the test mode.

Expected Results

If the processes perform successfully, the green color is displayed between each activity.

The InvokePartnerREST Project

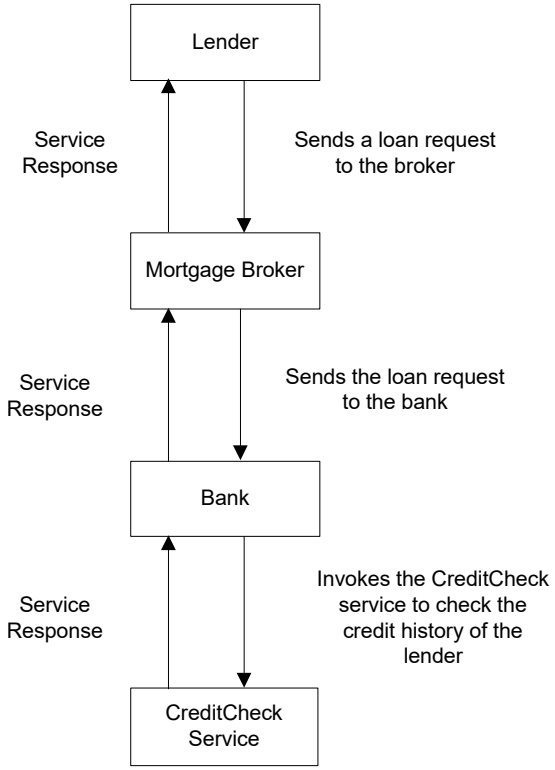
The Invoke_partner_rest project describes how to use the REST Dispatch and Reply activity to expose TIBCO ActiveMatrix BusinessWorks processes as RESTful web services and how to use the Invoke REST API activity to invoke the exposed RESTful web services.

Business Scenario

This project implements a complete workflow of asking for a loan from a bank through the mortgage broker as shown in Figure 19.

Here, a lender wants to ask for a loan from Bank C. The lender sends his request to the local broker. After receiving the loan request, the broker sends the loan request to Bank C that the lender prefers and Bank C invokes the CreditCheck service to check whether the lender is qualified for the loan.

Figure 19 Workflow of a Loan Request



Process Definition

Four process folders are included in the Invoke_partner_rest project:

- **Borrow** This folder contains one process, LoanRequestProcess, which is used by the lender to initiates a loan request.

  See LoanRequestProcess (in the Borrow Folder) for more details.

- **MortgageBroker** This folder contains a PartnerProcesses process folder and a LoanRequestProcess process for the broker to handle the request from the lender.

  — The LoanRequestProcess process is used for the broker to send the request to the corresponding bank according to the request from the lender.

  See LoanRequestProcess (in the MortgageBroker Folder) for more details.

  — The PartnerProcesses folder contains the partners of the broker. That is, banks that are available for the banker to invoke. The bank process in this folder invokes the corresponding bank service when a request is received.

  See PartnerProcess(in the MortgageBroker Folder) for more details.

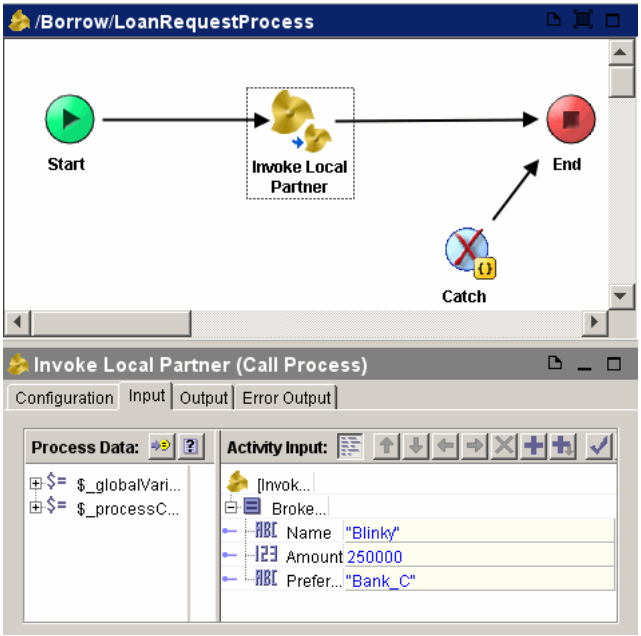
- **Banks** This folder contains two separate RESTful services to be invoked. Each folder contains one shared resource and three processes. See Banks for more details.

- **CreditCheck** This folder contains one shared resource and three processes. See Credit Check for more details.

LoanRequestProcess (in the Borrow Folder)

The LoanRequestRrocess process in the Borrow folder as shown in Figure 20, sends the loan request by invoking a local process.

Figure 20 The Lender Initiates a Loan Request

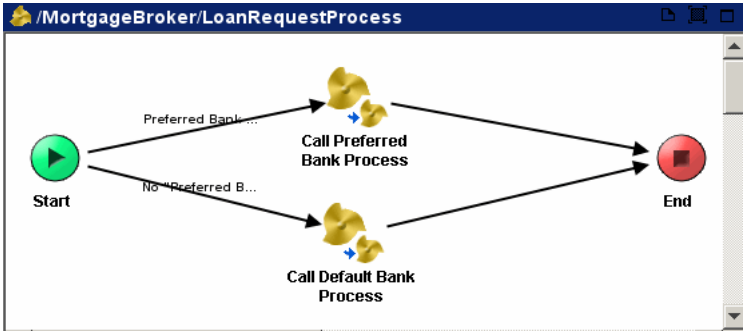


LoanRequestProcess (in the MortgageBroker Folder)

The LoanRequestProcess in the MortgageBroker folder as shown in Figure 21, sends the loan request to the corresponding bank based on the information provided by the lender.

- If the lender has a preferred bank, the CallPreferredBankProcess is run.
- If the lender has no preferred bank, the CallDefaultBankProcess is run.

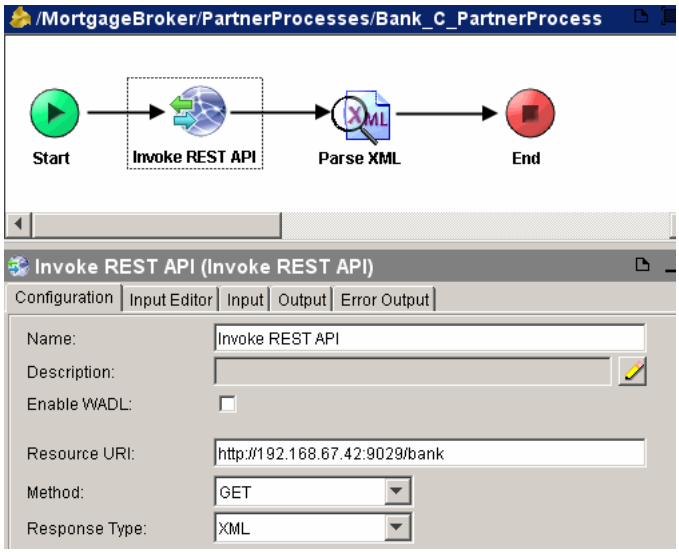
Figure 21 Sending the Request to the Bank



PartnerProcess(in the MortgageBroker Folder)

The loan request is sent to the corresponding Bank process in the PartnerProcess folder. Here, the lender chooses Bank_C, so the loan request is sent to Bank_C. As Figure 22 shows, Bank_C invokes its bank service to handle the loan request.

Figure 22 The Broker Sends the Loan Request to the Bank

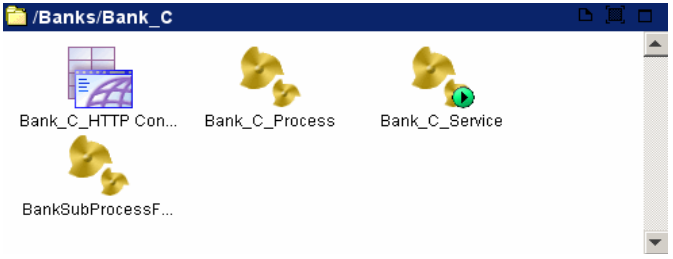


Banks

This folder contains two folders: Bank_A and Bank_C.

Take the Bank_C folder as an example. As shown in Figure 23, one shared resource and three processes are included in the Bank_C process folder.

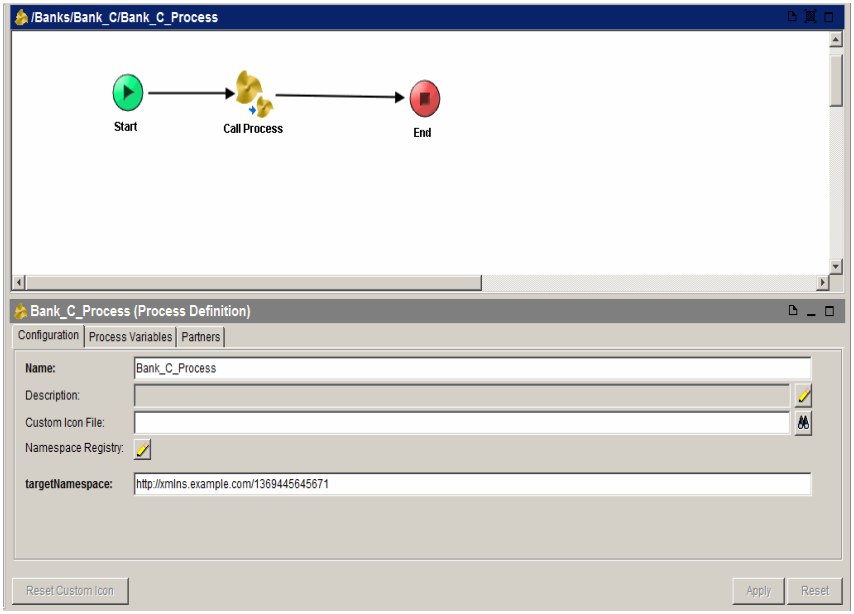
Figure 23 Bank_C Process Folder



- The Bank_C_HTTP_Connection shared resource is used to receive the incoming HTTP request.

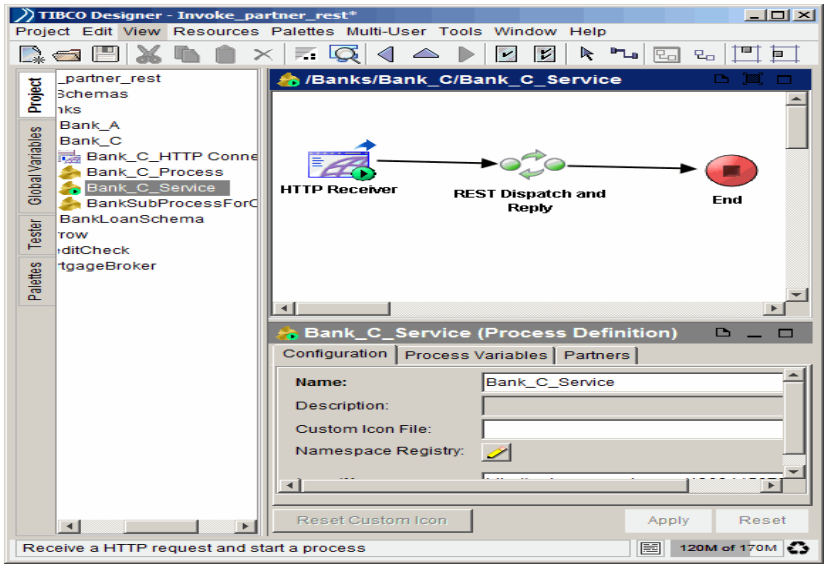
- The Bank_C_Process process is exposed as a RESTful web service. When a loan request is received, it invokes the subprocess to check the credit of the lender as shown in Figure 24.

Figure 24 The Bank_C_Process



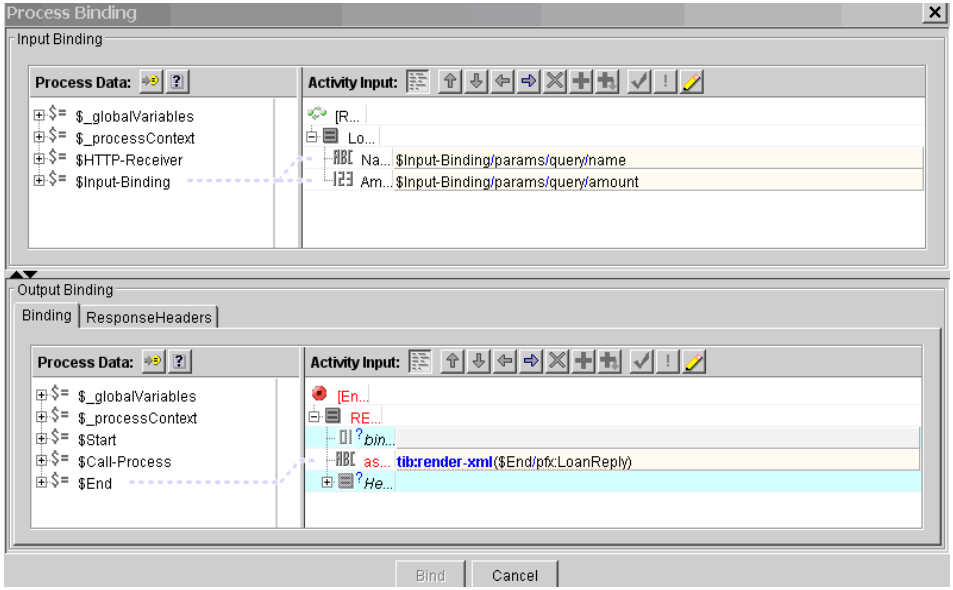
- The Bank_C_Service process is used to expose the Bank_C_Process process as a RESTful service as shown in Figure 25.

Figure 25 Bank_C_Service



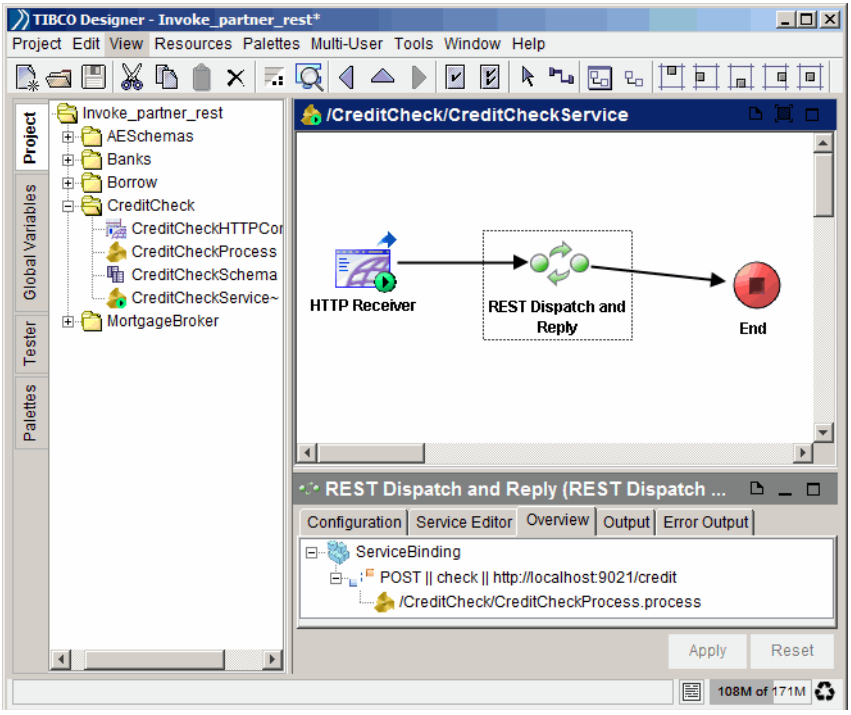
shows how the HTTP request and response are mapped to the input and output of the bound Bank_C_Process process.

Figure 26 Bank_C_Service



- The BankSubProcessForCreditCheck process is called by the Bank_C_Process process. The BankSubProcessForCreditCheck process invokes the CreditCheck service to check the credit history of the lender as shown in Figure 27.

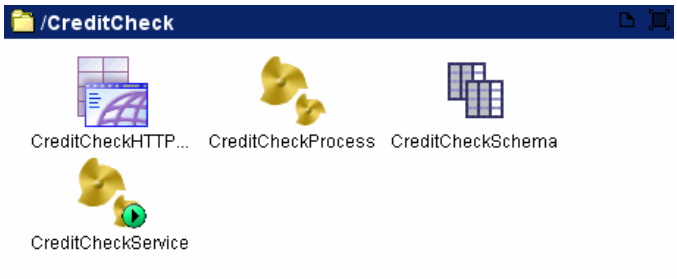
Figure 27 Invoking the CreditCheck Service



Credit Check

The CreditCheck folder includes one shared resource and two processes as shown in Figure 28.

Figure 28 Credit Check Folder



- The CreditCheckHTTPConnection shared resource is used to receive incoming HTTP request.
- The CreditCheckProcess process is exposed as a RESTful service. When the request of checking credit is received, it checks the credit of the account provided by the bank and sends the check result (approved or rejected) to the bank.
- The CreditCheckService process exposes the CreditCheckProcess process as a RESTful service as shown in Figure 29.

Figure 29 CreditCheck Service

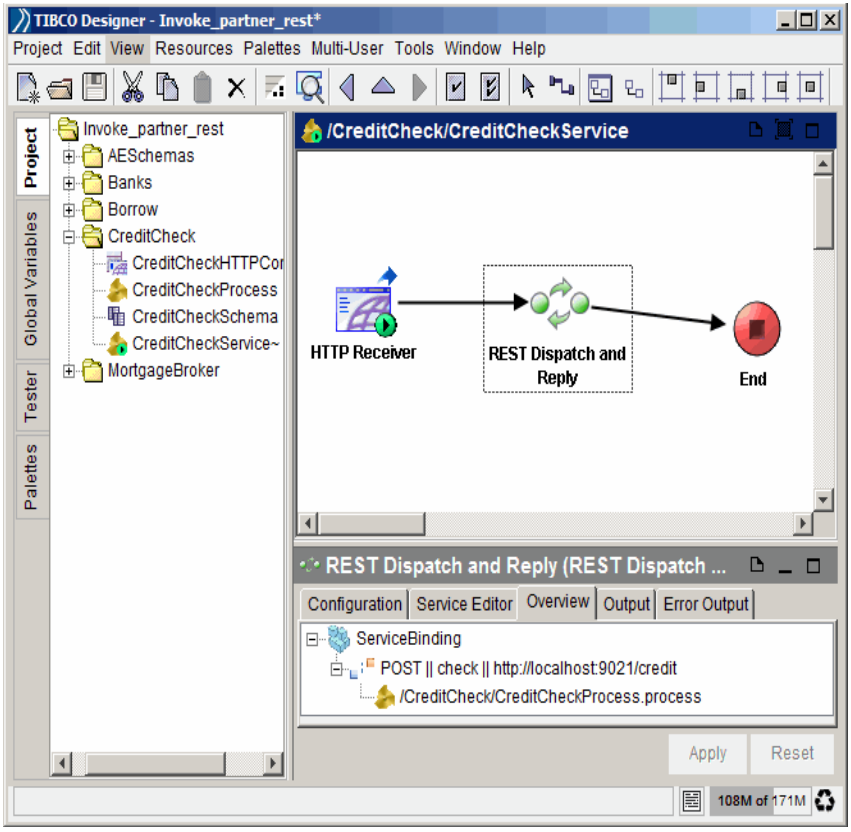
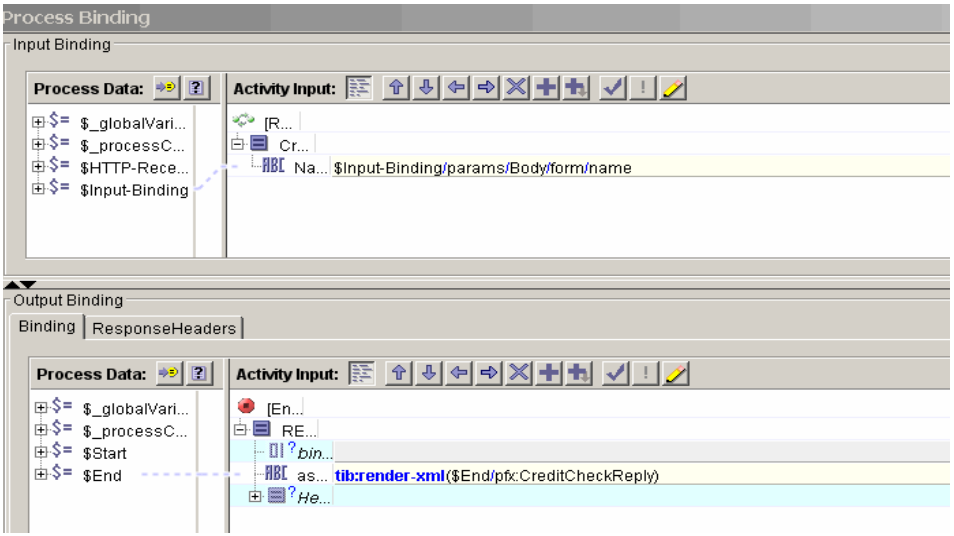


Figure 30 shows how the HTTP request and response are mapped to the input and output of the bound CreditCheck service process.

Figure 30 Credit Check Service Mapping

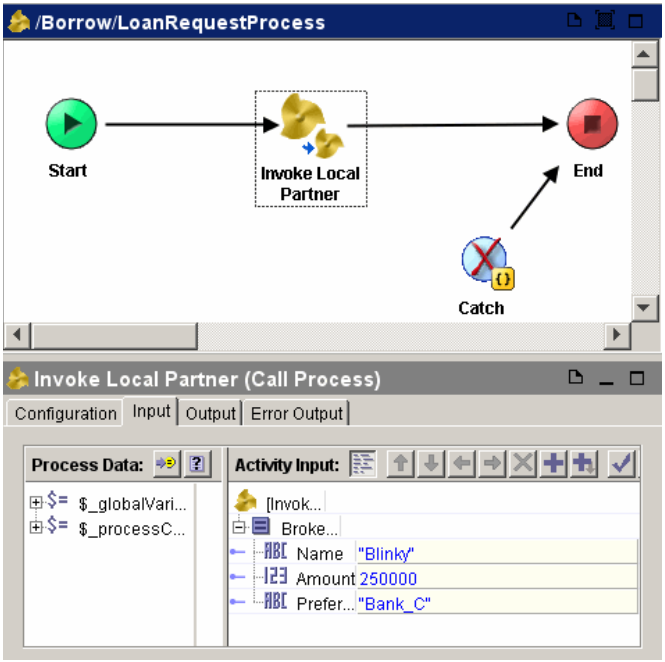


Setting Up the Project

Before running the project, you must perform the following steps to set up the project in TIBCO Designer:

1. Start TIBCO Designer.
2. Click **Open Existing Project** in the TIBCO Designer startup panel. The **Open Project** dialog is displayed.
3. Click **Browse** next to the **Project Directory** field, and then select the Invoke_partner_rest folder, which is located in the *RESTJSON_HOME*\examples directory. Click **OK**. The Invoke_partner_rest project is displayed.
4. Select **InvokePartnerREST > Borrow > LoanRequestProcess** in the Project panel.
5. Click the **InvokeLocalPartner** process and configure the **Input** tab as shown in Figure 31.

Figure 31 The Configuration Panel for the LoanRequestProcess



6. Select **Project > Save** from the TIBCO Designer menu to save the project.

Running the Project

After setting up the project, the next step is to trigger the processes in the InvokePartnerREST project:

1. In the Project panel, click the **Tester** tab, and then click **Start testing viewed process** at the top left corner in this panel. The **Select Processes To Load** dialog is displayed.
2. Click **Select All**, and then click **Load Selected** . The processes are loaded in the Tester panel.
3. Click the **Borrow/LoanRequestProcess.process** in the right panel, and then click to start running the process.

   When the process executes, the elements of the process change colors depending upon what is occurring in the executing process instance. If all the transition lines change to green, it means the process runs successfully.

4. Click **Stop testing** to return to Design mode.

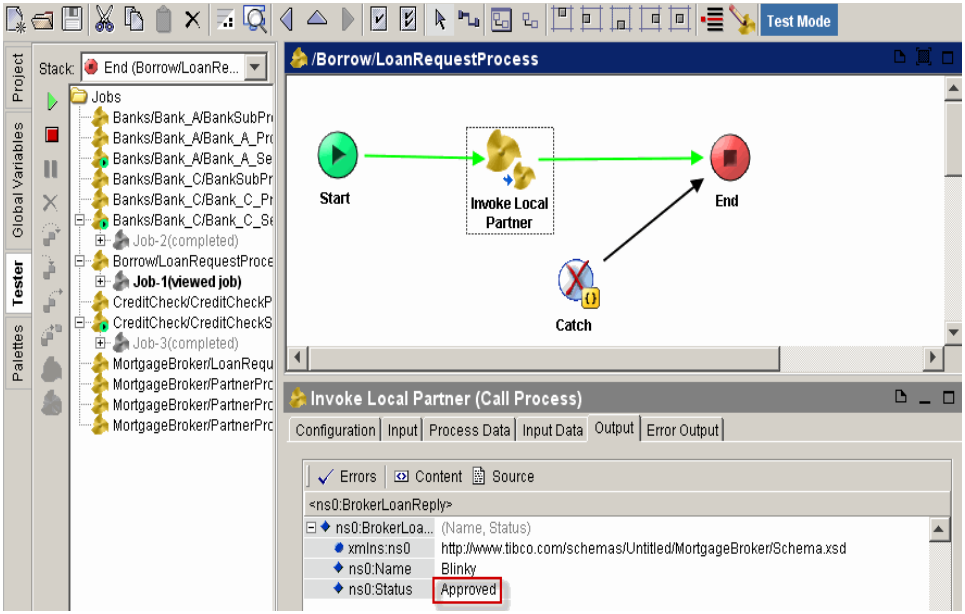
You can set breakpoints in the process definition at a specified point where you want to stop a running process and examine its state and process data, for example, you can set breakpoints before or after an activity.

See *TIBCO ActiveMatrix BusinessWorks™ Process Design Guide* for detailed information on using the test mode including setting breakpoints and the element colors in the test mode.

Expected Results

After the process is completed, the response is returned in the **Output** tab as shown in Figure 32.

Figure 32 Returned Response



Appendix A Advanced Topics

This chapter explains some advanced settings for use of TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON.

Topics

Implementation Mechanism Exchange for Parsing and Rendering

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports different implementation mechanisms when converting between JSON data and XML data based on the XSD.

Badgerfish conversion rule has the same impact when converting between JSON data and XML data in different implementation mechanisms.

This section uses an example to show the output of an implementation mechanism varies with the value of com.tibco.plugin.restjson.json.111CompatibleMode property. It also specifies how to select an implementation mechanism by configuring this property.

Example

Different implementation mechanisms affect the output when you convert data between JSON and XML format based on the XSD.

For example, when paring JSON data to XML data based on the provided XSD schemas, if you set the com.tibco.plugin.restjson.json.111CompatibleMode property to true, the attribute information in JSON is converted to a common XML element. Figure 33 shows an example of input data, and Figure 34 shows an example of output data.

Figure 33 JSON Input

```
{
    "root_person_info": {
        "age": "30",
        "person_id": "001a",
        "Information": {
            "Name": "Sarah",
            "Phone": {
                "Phone_attr": "Phone_information",
                "telephone": "8161000",
                "mobilephone": "15890980000"
            }
        },
        "employee": {
            "firstname_1": "Michael",
            "lastname_1": "Mannon",
            "address": "HappyValley",
            "city": "NewYork",
            "country": "USA"
        },
        "personinfo_child1": {
            "firstname_2": "Jim",
            "lastname_2": "Green"
        },
        "personinfo_child2": {
            "firstname_3": "Lucy",
            "lastname_3": "King"
        }
    }
}
```

Figure 34 Output Results When Property Is True

```
<root_person_info>
        <age>30</age>
        <person_id>001a</person_id>
        <Information>
                <Name>Sarah</Name>
                <Phone>
                        <Phone_attr>Phone_information</Phone_attr>
                        <telephone>8161000</telephone>
                        <mobilephone>15890980000</mobilephone>
                </Phone>
        </Information>
        <employee>
                <firstname_1>Michael</firstname_1>
                <lastname_1>Mannon</lastname_1>
                <address>HappyValley</address>
                <city>NewYork</city>
                <country>USA</country>
        </employee>
        <personinfo_child1>
                <firstname_2>Jim</firstname_2>
                <lastname_2>Green</lastname_2>
        </personinfo_child1>
        <personinfo_child2>
                <firstname_3>Lucy</firstname_3>
                <lastname_3>King</lastname_3>
        </personinfo_child2>
</root_person_info>
```

In the selected XSD file, "age" is defined as an attribute. When you set the property to false, the JSON pairs "age": "30" are converted to XML attributes and namespaces are added according to selected XSD file as shown in Figure 35.

Figure 35 Output Results When Property Is False

```
<ns1:root_person_info age = "30" xmlns:ns1 = "http://Namespace.com/Parent">
        <ns1:person_id>001a</ns1:person_id>
        <ns1:Information>
                <ns1:Name>Sarah</ns1:Name>
                <ns1:Phone Phone_attr = "Phone_information">
                        <ns2:telephone xmlns:ns2 = "http://Namespace.com/Child1">8161000</ns2:telephone>
                        <ns2:mobilephone xmlns:ns2 = "http://Namespace.com/Child1">15890980000</ns2:mobilephone>
                </ns1:Phone>
        </ns1:Information>
        <ns1:employee>
                <ns2:firstname_1 xmlns:ns2 = "http://Namespace.com/Child1">Michael</ns2:firstname_1>
                <ns2:lastname_1 xmlns:ns2 = "http://Namespace.com/Child1">Mannon</ns2:lastname_1>
                <ns1:address>HappyValley</ns1:address>
                <ns1:city>NewYork</ns1:city>
                <ns1:country>USA</ns1:country>
        </ns1:employee>
        <ns1:personinfo_child1>
                <ns2:firstname_2 xmlns:ns2 = "urn:child2">Jim</ns2:firstname_2>
                <ns2:lastname_2 xmlns:ns2 = "urn:child2">Green</ns2:lastname_2>
        </ns1:personinfo_child1>
        <ns1:personinfo_child2>
                <ns1:firstname_3>Lucy</ns1:firstname_3>
                <ns1:lastname_3>King</ns1:lastname_3>
        </ns1:personinfo_child2>
</ns1:root_person_info>
```

Selecting Implementation Mechanism

You can select an implementation mechanism supported in different product versions by configuring the com.tibco.plugin.restjson.json.111CompatibleMode property. The default value of this property is false indicating that the current version of implementation mechanism is adopted.

To select an implementation mechanism:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory.
2. Open the designer.tra file.
3. Add the com.tibco.plugin.restjson.json.111CompatibleMode property and set its value:
   - The true value indicates that the implementation mechanism in previous versions is adopted.
   - The false value indicates that the implementation mechanism in the current version is adopted.
4. Save the file and restart TIBCO Designer.

JSON Render Related Properties

When rendering XML data to JSON data, you can implement the following functions by configuring the JSON Render related properties:

- Checking Empty Elements for JSON Render
- Ignoring Namespace
- Escaping Forward Slash
- Formatting the Output String

Checking Empty Elements for JSON Render

If the XSD schema type is used when rendering the XML data to JSON data, by default, the XML elements that are not defined in XML but defined in XSD are ignored during the conversion.

However, such XML elements can be rendered to an empty JSON array or an empty JSON object accordingly by setting up a parameter, when the XSD schema type is used.

To preserve the empty XML elements when rendering JSON with the XSD schema type:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory.
2. Open the designer.tra file.
3. Add the com.tibco.plugin.restjson.xml2json.checkEmptyElement property in the file and set its value to true.
4. Save the file and restart TIBCO Designer.

If such elements have a namespace defined, after conversion, the namespace with emp followed after the prefix xmlns is added to the JSON objects.

Example

For example, the input data is shown as follows:

```
<home>
 <address>
  <address2>homeAddress</address2>
  <city>cityName</city>
  <state>StateName</state>
 </address>
</home>
```

After you set the property value to true, the empty XML elements that are not defined in XML but defined in XSD are rendered into empty JSON array or empty JSON objects when the XSD schema type is used. The output is displayed as follows:

```
{
 "home":{
  "address":{
   "address1":[
   ],
   "address2":"homeAddress",
   "city":"cityName",
   "state":"StateName"
  }
 }
}
```

If you set the property value to false or do not add the property, the output is displayed as follows:

```
{
 "home":{
  "address":{
   "address2":"homeAddress",
   "city":"cityName",
   "state":"StateName"
  }
 }
}
```

Ignoring Namespace

You can configure the com.tibco.plugin.restjson.xml2json.namespaceIgnored property to ignore the namespace and prefix information when converting XML to JSON. The default value is false, indicating that the namespace information is considered when rendering JSON files. The value true indicates that the namespace information is ignored.

This property is applicable for the Generic and XSD schema types.

To ignore the namespace information:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory.
2. Open the designer.tra file.
3. Add the com.tibco.plugin.restjson.xml2json.namespaceIgnored property in the file and set its value to true.

4. Save the file and restart TIBCO Designer.

Example

For example, the input data is shown as follows:

```
<root_ns xmlns = "http://NamespaceTest.com/Parent">
 <person_id>001a</person_id>
</root_ns>
```

After you set the property value to true, the namespace is ignored. The output data is displayed as follows:

```
{
 "root_ns":{
  "person_id":{
   "$":"001a"
  }
 }
}
```

If you do not add this property, or set this property value to false, the output is displayed as follows:

```
{
 "root_ns":{
  "@xmlns":"http://NamespaceTest.com/Parent",
  "person_id":{
   "$":"001a"
  }
 }
}
```

Escaping Forward Slash

You can configure the com.tibco.plugin.restjson.xml2json.forwardSlashEscaping property to escape the forward slash when rendering JSON files. The default value is false, indicating that the forward slash is kept when rendering JSON files. The value true indicates that the forward slash is escaped: "/" is converted to "\/".

This property is applicable for the Generic and XSD schema types.

To escape the forward slash:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory.
2. Open the designer.tra file.

3. Add the com.tibco.plugin.restjson.xml2json.forwardSlashEscaping property in the file and set its value to true.
4. Save the file and restart TIBCO Designer.

Example

For example, the input data is shown as follows:

```
<Phone>
 <telephone xmlns =  "http://NamespaceTest.com/test">8161000</telephone>
</Phone>
```

After you set the property value to true, the forward slash is escaped. The output data is displayed as follows:

```
{
 "Phone":{
  "telephone":{
   "@xmlns":"http:\/\/NamespaceTest.com\/test",
   "$":"8161000"
  }
 }
}
```

If you do not add this property, or set this property value to false, the output is displayed as follows:

```
{
 "Phone":{
  "telephone":{
   "@xmlns":"http://NamespaceTest.com/test",
   "$":"8161000"
  }
 }
}
```

Formatting the Output String

You can configure the com.tibco.plugin.restjson.json.prettyPrint property to decide whether you want to format the output string. The default value is true, indicating that you want to format the output string.

This property is applicable for the Generic and XSD schema types.

To format the output string:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory.

2. Open the designer.tra file.
3. Add the com.tibco.plugin.restjson.json.prettyPrint property in the file and set its value to true.
4. Save the file and restart TIBCO Designer.

Example

For example, the input data is shown as follows:

```
<root_attr attr_default="newDefault" attr_fixed="attr_fixed_value">
 <people_info>
  <people_address attr_optional="attr_required_optional" attr_required="attr_required_value">
   <street>XD</street>
   <city>BJ</city>
   <state>BJ</state>
   <country>China</country>
  </people_address>
  <children_number>35</children_number>
 </people_info>
 <product prodid="12">tv</product>
 <shoesize color="black">39</shoesize>
</root_attr>
```

If you set the property value to true or do not add this property, the output data is displayed as follows:

```
{
 "root_attr": {
  "@attr_default": "newDefault",
  "@attr_fixed": "attr_fixed_value",
  "people_info": {
   "people_address": {
       "@attr_optional": "attr_required_optional",
       "@attr_required": "attr_required_value",
       "street": {
        "$": "XD"
       },
       "city": {
        "$": "BJ"
       },
       "state": {
        "$": "BJ"
       },
```

```
        "country": {
          "$": "China"
        }
      },
      "children_number": {
        "$": 35
      }
    },
    "product": {
      "@prodid": 12,
      "$": "tv"
    },
    "shoesize": {
      "@color": "black",
      "$": 39
    }
  }
}
```


If you set this property value to false, the output is displayed as follows:


```
{"root_attr":{"@attr_default":"newDefault","@attr_fixed":"attr_fixed_value","people_info":{"people_address":
{"@attr_optional":"attr_required_optional","@attr_required":"attr_required_value","street":{"$":"XD"},"city":{"$":"B
J"},"state":{"$":"BJ"},"country":
{"$":"China"}},"children_number":{"$":35}},"product":{"@prodid":12,"$":"tv"},"shoesize":{"@color":"black","$":39
}}}
```

JSON Parse Related Properties

When parsing JSON data to XML data, you can implement the following functions by configuring the JSON Parse related properties:

- Adding JSON root
- Converting the Special Characters to ASCII Code
- Preferring Namespace from Schema Using StAXON

Adding JSON root

You can configure the com.tibco.plugin.restjson.json2xml.jsonroot property to add the JSON root name. If you do not set the property, <JSON> is used as the default root. This property is only applicable for the Generic schema type.

To add the JSON root:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory.
2. Open the designer.tra file.
3. Add the com.tibco.plugin.restjson.json2xml.jsonroot property in the file and set the root value.
4. Save the file and restart TIBCO Designer.

Example

For example, the input data is shown as follows:

{"product":"REST","name":"json"}

When you set the root name, for example, JsonRootName, the output data is displayed as follows:

```
<JsonRootName>
 <product>REST</product>
 <name>json</name>
</JsonRootName>
```

If you do not add this property, or do not assign a value to this property, JSON is used as the default root name.

```
<JSON>
 <product>REST</product>
 <name>json</name>
</JSON>
```

Converting the Special Characters to ASCII Code

When you parse the JSON files, the special character is ignored by default. If you want to convert the special characters to ASCII code, you can configure the com.tibco.plugin.restjson.json2xml.specialIgnored property. This property is applicable for the Generic and the XSD schema types.

To convert the special characters to ASCII code:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory.
2. Open the designer.tra file.
3. Add the com.tibco.plugin.restjson.json2xml.specialIgnored property in the file and set its value to false.
4. Save the file and restart TIBCO Designer.

Example

For example, the input data is shown as follows:

```
{"Product":{"Nam$e":"REST"}}
```

If you do not add this property or set the value to true, the output data is displayed as follows:

```
<Product>
 <Name>REST</Name>
</Product>
```

If set the value to false, the output is displayed as follows:

```
<Product>
 <Nam_-36-_e>REST</Nam_-36-_e>
</Product>
```

Preferring Namespace from Schema Using StAXON

You can configure the com.tibco.plugin.restjson.json2xml.preferNamespaceFromSchema Boolean property to prefer namespace from schema. If you do not set this property to true, the default value is false.

To prefer namespace from schema using StAXON:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory.
2. Open the designer.tra file.
3. Add the com.tibco.plugin.restjson.json2xml.preferNamespaceFromSchema property in the file and set the boolean value.
4. Save the file and restart TIBCO Designer.

Example

For example, the input data is shown as follows:

```
{"user_cred": {"user_name": "ab12345", "user_password": "45678"}}
```

Output editor root element: 'user_root'

Default namespace from schema: http://www.tibco.com/schemas/TestStaxon/Schema.xsd

When property com.tibco.plugin.restjson.json2xml.preferNamespaceFromSchema is set to true, the output data is displayed as follows:

```
<user_root xmlns = "http://www.tibco.com/schemas/TestStaxon/Schema.xsd">
 <user_cred>
  <user_name>ab12345</user_name>
  <user_password>45678</user_password>
 </user_cred>
</user_root>
```

When property com.tibco.plugin.restjson.json2xml.preferNamespaceFromSchema is set to false, the output data is displayed as follows:

```
<user_root>
 <user_cred>
  <user_name>ab12345</user_name>
  <user_password>45678</user_password>
 </user_cred>
</user_root>
```

Setting Up the Thread Pool

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON employs a thread pool to execute all the sending HTTP request operations, which is very efficient with a high degree of concurrency.

The number of threads in the pool determines the number of operations that can simultaneously be executed. The larger the thread-count, the more computer resources have to be used to maintain and switch contexts between threads.

The default value for the thread pool is 10. To set up a larger value for the thread pool, do the following:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory and locate the designer.tra file.
2. Open the designer.tra file.
3. Set up a larger value for the bw.plugin.http.client.ResponseThreadPool parameter.
4. Save the file.

After configuring the configuration file, restart TIBCO Designer.

Setting Up a Proxy

TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON supports invoking REST APIs by using proxy.

To specify a proxy for use, complete the following steps:

1. Navigate to the *TIBCO_HOME*\designer*version_number*\bin directory and locate the designer.tra file.
2. Open the designer.tra file.
3. Add the proxy parameters listed in Table 34.
4. Save the file.

Table 34 Proxy Parameters

Parameter	Description
com.tibco.plugin.restjson.proxyHost	The IP address of the machine that you want to use as the proxy.
com.tibco.plugin.restjson.proxyPort	The port number of the proxy machine.
com.tibco.plugin.restjson.proxyUser	The account used to access to the proxy machine.
com.tibco.plugin.restjson.proxyPwd	The password for the account used to access to the proxy machine.

Support for SSL Configuration

TIBCO ActiveMatrix BusinessWorks Plug-in for REST and JSON supports invoking a Service Provider with SSL configurations.

The Invoke REST API activity uses the Java key store for SSL. Java key store ships certificates signed by a well-known certification authority (CA), such as, VeriSign. Hence, no configuration is needed for the Invoke REST API activity if the Service Provider uses the certificates from CA.

If the Service Provider does not use well-known CA certificates or requires a specific certificate, perform the following steps:

- Import the intermediate and root certificates to the *TIBCO_HOME*\tibcojre*version_number*\lib\security\cacerts directory.
- If the REST application requires a specific authentication other than the typical OAuth authentication, you must request a client certificate from the same CA as that for your REST application and then import the certificate to the *TIBCO_HOME*\tibcojre*version_number*\lib\security\cacerts directory.
- You can create a folder in the location where you store the trusted certificates, and then create a BW_GLOBAL_TRUSTED_CA_STORE global variable in your TIBCO ActiveMatrix BusinessWorks project to point to this folder.

  The value you set for BW_GLOBAL_TRUSTED_CA_STORE must be a file URL, for example, file:///c:/tibco/certs. See the information about trusted certificates in *TIBCO ActiveMatrix BusinessWorks™ Process Design* for details about how to store certificates to your project.

Support for Cross-Origin Requests

When you set up a REST server where the REST Dispatch and Reply activity is located, and set up another web application server containing all the web components, such as HTML, CSS, and JavaScript, a cross-origin request is initiated when you use a browser visiting the web application server to call RESTful web services exposed by the REST server.

This section specifies several rules that apply to the REST Dispatch and Reply activity when handling cross-origin requests.

Ordinary Requests

The REST Dispatch and Reply activity handles cross-origin requests according to the following rules:

- The Value of the origin parameter in the request header maps to the value of the Access-Control-Allow-Origin parameter in the response header.
- The value of the Access-Control-Allow-Methods parameter is set to OPTIONS, GET, PUT, POST, DELETE, and *.

You can add more headers if needed. Additionally, you can override the default values of these two headers by adding headers with the same name.

Preflight Requests

The REST Dispatch and Reply activity handles OPTIONS requests according to the following rules:

- The value of the origin parameter in the request header maps to the value of the Access-Control-Allow-Origin parameter in the response header.
- The value of the access-control-request-method parameter in the request header maps to the value of the Access-Control-Allow-Methods parameter in the response header.
- The value of the access-control-request-headers parameter in the request header maps to the value of the Access-Control-Allow-Headers parameter in the response header.
- By default, the value of the Access-Control-Max-Age parameter in the response header is set to 20 days. You can add the same response headers to overwrite default values.

Web browsers have different implementation mechanisms. Normally, the Access-Control-Max-Age parameter is expired before the value set in it.

Potential Failures and Solutions

The REST Dispatch and Reply activity might fail to handle cross-origin requests because of an incorrect activity configuration, difference in the versions of TIBCO ActiveMatrix BusinessWorks and TIBCO Runtime Agent, or use of different browsers.

TIBCO Runtime Agent Versions

The REST Dispatch and Reply activity fails to handle cross-origin requests when using the TIBCO Runtime Agent version 5.7.x or earlier. To handle cross-origin requests TIBCO Runtime Agent 5.8.0 or later is installed.

Different Server Types

When you use the HTTP Receiver activity to receive a HTTP request, if the server uses Tomcat to handle cross-origin requests, an error occurs. Ensure that the server uses HTTP Component to handle cross-origin requests.

There is an exception that the HTTP Receiver activity can handle cross-origin requests, even though the server uses Tomcat. The prerequisite is that you must install TIBCO Runtime Agent 5.9 or above and TIBCO ActiveMatrix BusinessWorks 5.12 or above. Then, add the bw.plugin.http.enableDoOptions=true property to the designer.tra file in the *TIBCO_HOME*\designer*version_number*\bin directory.

Appendix B Trace Messages

This appendix explains the trace messages used by TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON.

Topics

Overview of Trace Messages

Trace messages provide information about plug-in activities. The messages are logged to a log file and the console where the runtime plug-in is started.

Introduction of Trace Message Roles

The roles of trace messages in TIBCO ActiveMatrix BusinessWorks™ Plug-in for REST and JSON are shown in Table 35.

Table 35 Trace Message Roles

Role Name	Description
Info	Indicates normal plug-in operation. No action is necessary. A tracing message tagged with Info. indicates that a significant processing step is reached and logged for tracking or auditing purposes. Only info. messages preceding a tracking identifier are considered significant steps.
Warn	An abnormal condition is found. Processing will continue, but special attention from an administrator is recommended.
Error	An unrecoverable error occurs. Depending on the error severity, the adapter may continue with the next operation or may stop altogether.
Debug	A developer-defined tracing message. In normal operating conditions, debug messages should not display.

Setting Custom Engine Properties for Trace Message Roles

The following sections describe how to set properties of trace message roles in testing environment and in deployed projects:

- Custom Properties in Testing Environment
- Custom Properties in Deployed Projects

Each property represents a trace message role as shown in Table 36. The default value for the trace message roles are shown in the Default Value column. If the property file is not defined, the default value will be used for the message roles.

Table 36 Properties of Trace Message Roles

Property	Trace Message Role	Default Value
Trace.Info.*	Info	True
Trace.Error.*	Error	True
Trace.Warn.*	Warn	True
com.tibco.plugin.restjson.debug	Debug	False

Custom Properties in Testing Environment

To enable or disable a trace message role in testing environment, you need to set custom property and to specify its location. Perform the following tasks to set properties for trace message roles in testing environment:

- Task A, Create and Configure a Property File
- Task B, Specify the Properties in Testing Environment

Task A Create and Configure a Property File

Perform the following steps to create and configure a property file:

1. Create a property file in the directory where you want to save the file, and specify a name for the file. For example, properties.cfg.
2. Set the properties in the created file to configure the trace message roles. See Table 36 for the property of each message role.

To set property values, follow the rules:

— To enable a trace message role, set the value of the corresponding property to true. For example, the following line sets the Trace.Info.* property to true:

```
Trace.Info.*=true
```

— To disable a trace message role, set the value of the corresponding property to false. For example, the following line sets the Trace.Error.* property to false:

```
Trace.Error.*=false
```

Task B Specify the Properties in Testing Environment

Perform the following steps to set properties when testing a process:

1. Start TIBCO Designer and then open the project you want to specify the test.
2. Select **Tools > Tester > Start** from the TIBCO Designer menu. The **Select Processes to Load** dialog is displayed.
3. Specify the property file you created in Task A.
   a. Click **Advanced** in the **Select Processes to Load** dialog. The **Advanced Test Settings** dialog is displayed.
   b. In the **Test Engine User Args** field, enter the following values:

      -p *file_directory*/*file_name*

      where,

      file_directory is the location of the property file name, for example, c:/tibco.

      file_name is the property file name, for example, properties.cfg.

   c. Click **OK** to save the setting.

The enabled messages are logged to the console where the runtime plug-in was started. See "Custom Engine Properties" in *TIBCO ActiveMatrix BusinessWorks™ Administration* for more information.

Custom Properties in Deployed Projects

TIBCO Administrator is responsible for deploying process engines in a production environment. TIBCO ActiveMatrix BusinessWorks™ provides a file, bwengine.xml, for specifying custom properties in deployed engines. The bwengine.xml file is located in the *TIBCO_HOME*\bw*release_version*\lib\com\tibco\deployment directory.

The file has a properties element that defines all of the properties in the deployed process engine. Each property is contained in a property element with the following structure:

```
<property>
 <name>name to display in TIBCO Administrator</name>
 <option>name of property</option>
 <default>default value</default>
 <description>short description of property</description>
</property>
```

Perform the following steps to set properties for trace message roles in deployed projects:

1. Open the bwengine.xml file.
2. Add properties in the file to configure the trace message roles. For example, the following lines set the com.tibco.plugin.restjson.debug property to true:

```
<property>
 <name>RESTJSON Plug-in Debug Role</name>
 <option>com.tibco.plugin.restjson.debug</option>
 <default>true</default>
 <description>Indicates RESTJSON debug message property.</description>
</property>
```

   After defining the property in the bwengine.xml file, it is available in Enterprise Archive Files that are created by TIBCO Designer, and it is displayed in the **Advanced** tab of the deployment configuration in TIBCO Administrator.

3. Save EAR files in TIBCO Designer and load them into any deployment configurations created in TIBCO Administrator.

   You can alter the value of a property in the **Advanced** tab of the deployment configuration and that value will be used in the deployed project. For more information about creating and managing deployment configurations, see "Creating and Deploying Applications" in *TIBCO ActiveMatrix Businessworks™ Administration*.

For more information about setting properties in deployed projects, see "Setting Custom Engine Properties in Deployed Projects" in *TIBCO ActiveMatrix Businessworks™ Administration*.

Error Codes

BW-RESTJSON-000025: Start to operate [%1]. Process Id: [%2]. Engine Name: [%3].

Role: debugRole

Category: BW_Plug-in

Description: Indicates the started operation information.

Resolution: No action required.

BW-RESTJSON-000026: The end of the [%1] activity.

Role: debugRole

Category: BW_Plug-in

Description: The operation of the activity has finished.

Resolution: No action required.

BW-RESTJSON-100013: An error occurs while validating output: %1.

Role: errorRole

Category: BW_Plug-in

Description: The output data is invalid.

Resolution: Check whether the XML data matches the schema file.

BW-RESTJSON-100014: An error occurs in render activity: %1.

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when running the Render activity.

Resolution: Depends on the content of the exception.

BW-RESTJSON-100016: An error occurs while converting json to xml: %1.

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when converting JSON to XML.

Resolution: Depends on the content of the exception.

BW-RESTJSON-100021: JSON Schema Tool- Start to generate XML schema from JSON file, the JSON file is [%1].

Role: infoRole

Category: BW_Plug-in

Description: Indicates the XML schema has been started to generate.

Resolution: No action required.

BW-RESTJSON-100022: JSON Schema Tool- Schema file generated successfully under [%1] folder.

Role: infoRole

Category: BW_Plug-in

Description: Indicates the schema file has been generated successfully.

Resolution: No action required.

BW-RESTJSON-100023: JSON Schema Tool- Failed to generate schema file due to [%1].

Role: infoRole

Category: BW_Plug-in

Description: Indicates the reason why the schema file has not been generated.

Resolution: No action required.

BW-RESTJSON-100025: Parse operation with type [%1].

Role: debugRole

Category: BW_Plug-in

Description: Indicates the information about the parse schema type.

Resolution: No action required.

BW-RESTJSON-100026: Start to convert JSON to XML.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the conversion from JSON to XML has been started.

Resolution: No action required.

BW-RESTJSON-100027: The JSON message to be parsed is: %1.

Role: debugRole

Category: BW_Plug-in

Description: Provides additional runtime information where %1 is the specified JSON message.

Resolution: No action required.

BW-RESTJSON-100028: Validate output is enabled:[%1].

Role: debugRole

Category: BW_Plug-in

Description: Indicates whether the Validate Output option is enabled or not.

Resolution: No action required.

BW-RESTJSON-100029: The generated XML is: %1.

Role: debugRole

Category: BW_Plug-in

Description: Provides additional runtime information where %1 is the specified XML message.

Resolution: No action required.

BW-RESTJSON-100030: Parse operation executed successfully.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the parse operation executed successfully.

Resolution: No action required.

BW-RESTJSON-100031: Remove root option is enabled:[%1].

Role: debugRole

Category: BW_Plug-in

Description: Indicates whether the Remove Root option is enabled or not.

Resolution: No action required.

BW-RESTJSON-100032: Render operation with type [%1].

Role: debugRole

Category: BW_Plug-in

Description: Indicates the information about the render schema type.

Resolution: No action required.

BW-RESTJSON-100033: Badgerfish conversion is in use:[%1].

Role: debugRole

Category: BW_Plug-in

Description: Indicates whether the Badgerfish conversion is enabled or not.

Resolution: No action required.

BW-RESTJSON-100034: Start to convert XML to JSON.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the conversion from XML to JSON has started.

Resolution: No action required.

BW-RESTJSON-100035: The XML to be rendered is: %1

Role: debugRole

Category: BW_Plug-in

Description: Provides additional runtime information where %1 is the specified XML message.

Resolution: No action required.

BW-RESTJSON-100036: The generate JSON data is: %1.

Role: debugRole

Category: BW_Plug-in

Description: Provides additional runtime information where %1 is the generated JSON data.

Resolution: No action required.

BW-RESTJSON-100037: Render operation executed successfully.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the render operation executed successfully.

Resolution: No action required.

BW-RESTJSON-100038: The DataTypeFactory for java type [javax.xml.datatype.Duration] is not provided, the converter will not claim the responsibility for Duration objects.

Role: warnRole

Category: BW_Plug-in

Description: The DataTypeFactory for Java type was not provided.

Resolution: No action required.

BW-RESTJSON-100039: Validate input is enabled: [%1].

Role: debugRole

Category: BW_Plug-in

Description: Indicates the Validate Input option has been enabled.

Resolution: No action required.

BW-RESTJSON-200001: Operation with WADL enabled, the path of WADL file is [%1].

Role: debugRole

Category: BW_Plug-in

Description: Indicates an operation with WADL has been enabled.

Resolution: No action required.

BW-RESTJSON-200002: Operation with WADL disabled.

Role: debugRole

Category: BW_Plug-in

Description: Indicates an operation with WADL has been disabled.

Resolution: No action required.

BW-RESTJSON-200003: Basic configuration, the request method is: %1, authentication style is: [%2].

Role: debugRole

Category: BW_Plug-in

Description: Indicates the basic configuration information about the Invoke REST API activity.

Resolution: No action required.

BW-RESTJSON-200004: The response type of GET request is [%1].

Role: debugRole

Category: BW_Plug-in

Description: Indicates the response type of the get method.

Resolution: No action required.

BW-RESTJSON-200005: The encoded request URL is %1.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the encoded request URL.

Resolution: No action required.

BW-RESTJSON-200006: %1 request with body content in it.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the request was sent with body content. The %1 is one of the four REST methods: GET, POST, PUT, and DELETE.

Resolution: No action required.

BW-RESTJSON-200007: %1 request without body content in it.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the request was sent without body content.

Resolution: No action required.

BW-RESTJSON-200008: REST request executed successfully.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the REST request has been executed successfully.

Resolution: No action required.

BW-RESTJSON-2000011: Invoke REST API with proxy enabled, ip: [%1], port: [%2]

Role: debugRole

Category: BW_Plug-in

Description: Indicates the proxy is enabled when invoking a REST API.

Resolution: No action required.

BW-RESTJSON-2000012: Rich output enabled.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the Rich Output option is selected.

Resolution: No action required.

BW-RESTJSON-2000013: Trust HTTPS host enabled, the program will not validate the host.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the Trust HTTPS Host option is selected and the host will not be validated.

Resolution: No action required.

BW-RESTJSON-2000014: Use customer defined timeout: [%1]ms.

Role: debugRole

Category: BW_Plug-in

Description: Indicates the specified timeout value for the HTTP response.

Resolution: No action required.

BW-RESTJSON-2000019: Error happened when invoking REST server, [%1].

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when invoking the REST server.

Resolution: The REST server error.

BW-RESTJSON-2000020: Both the Resource URI and Method are required, make sure they are not empty.

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when running the Invoke REST API activity.

Resolution: Ensure that the specified Resource URI, and Method fields in the **Configuration** tab are not empty.

BW-RESTJSON-2000021: Fail to encode the URI, the operating system doesn't support UTF-8.

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when running the Invoke REST API activity with the UTF-8 encoding character in the **URI** field.

Resolution: Change the encoding character in the URI field.

BW-RESTJSON-2000022: Authentication params error: [%1].

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when running the Invoke REST API activity.

Resolution: Ensure that all the values are provided to the required Authentication parameters.

BW-RESTJSON-2000023: Try to add OAuth2 parameters to a position other than 'Query' and 'Header'.

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when setting the OAuth2 parameters to the incorrect input items.

Resolution: Ensure the OAuth2 parameters are set to the Query and Header items in the **Input** tab.

BW-RESTJSON-2000024: Meet an illegal body part.

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when using the unsupported items in the body of the HTTP request message.

Resolution: Ensure that only the Form, Text, and Binary items are defined in the body of the HTTP request message.

BW-RESTJSON-2000025: Internal error occurs: %1.

Role: errorRole

Category: BW_Plug-in

Description: This is an internal error.

Resolution: Depends on the content of the exception.

BW-RESTJSON-3000001: Internal Server error occurs: %1.

Role: errorRole

Category: BW_Plug-in

Description: An error occurs in the server side.

Resolution: Check your configurations.

BW-RESTJSON-3000002: =Error!==========> %1

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when binding the TIBCO ActiveMatrix BusinessWorks process.

Resolution: Check your configurations.

BW-RESTJSON-400000: =Export Swagger. Start to export Swagger, the Swagger file is [%1].

Role: infoRole

Category: BW_Plug-in

Description: Indicates the Swagger API description file is started to export.

Resolution: No action required.

BW-RESTJSON-400001: =Export Swagger. Swagger file is exported successfully.

Role: infoRole

Category: BW_Plug-in

Description: Indicates the Swagger API description file is exported successfully.

Resolution: No action required.

BW-RESTJSON-400002: =Export Swagger. Failed to export Swagger file due to [%1].

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when exporting Swagger API description file.

Resolution: Check your configurations.

BW-RESTJSON-400003: =Export Swagger. There are [%1] resources in the application, and resource number [%2] is just exported to Swagger.

Role: infoRole

Category: BW_Plug-in

Description: Indicates the amount of the resource in the application is exported to Swagger.

Resolution: No action required.

BW-RESTJSON-500000: =Export WADL. Start to export WADL, the WADL file is [%1].

Role: infoRole

Category: BW_Plug-in

Description: Indicates the WADL file is started to export.

Resolution: No action required.

BW-RESTJSON-500001: =Export WADL. The WADL file is exported successfully.

Role: infoRole

Category: BW_Plug-in

Description: Indicates the WADL file is exported successfully.

Resolution: No action required.

BW-RESTJSON-500002: =Export WADL. Failed to export WADL due to [%1].

Role: errorRole

Category: BW_Plug-in

Description: An error occurs when exporting the WADL file.

Resolution: Check your configurations.
