Important Information

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

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

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

TIBCO, TIBCO ActiveMatrix BusinessWorks, TIBCO Rendezvous, TIBCO Enterprise Message Service, TIBCO Business Studio, TIBCO Enterprise Administrator, TIBCO ActiveSpaces, TIBCO Runtime Agent, TIBCO Designer, and Two-Second Advantage are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

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

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

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

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

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

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

Copyright © 2001-2016 TIBCO Software Inc. All rights reserved.

TIBCO Software Inc. Confidential Information
## Contents

**Figures** ................................................................. 6  
**TIBCO Documentation and Support Services** .................................................. 7  
**What is REST?** ................................................................. 8  
  - What is a Resource? .................................................. 8  
  - What are Operations? .................................................... 8  
  - What is an Endpoint? ..................................................... 8  
**What is Swagger?** ................................................................. 10  
  - The Swagger UI .......................................................... 10  
    - Viewing Multiple REST Services in Swagger UI .......................................................... 10  
**Parameter Support** ................................................................. 11  
  - Conversion Between Swagger and XML in TIBCO Business Studio .......................................................... 11  
  - Working with Path and Query Parameters .......................................................... 18  
    - Adding or Editing Path Parameters .......................................................... 19  
    - Adding or Editing Query Parameters .......................................................... 20  
  - Working with Arrays .......................................................... 20  
  - Working with Form Parameters .......................................................... 23  
    - Creating Form Parameters .......................................................... 23  
  - Working with Header Parameters .......................................................... 24  
**About REST Services and References** ................................................................. 26  
**Supported Message Formats** ................................................................. 27  
**Restrictions** ................................................................. 28  
  - Restrictions on JSON .......................................................... 28  
  - Restrictions on XML Schema .......................................................... 28  
**Using Swagger 1.2 Files** ................................................................. 29  
**REST Schema Utilities** ................................................................. 30  
  - Creating a New XML Schema File .......................................................... 30  
  - Creating XML Schema From a JSON Payload .......................................................... 30  
  - Generating an XML Schema from a Swagger File .......................................................... 31  
**REST Service** ................................................................. 32  
  - REST Service Binding .......................................................... 32  
  - Creating a REST Service .......................................................... 35  
    - Using Swagger to Create a REST Service .......................................................... 35  
    - Using the Wizard to Create a REST Service .......................................................... 35  
    - Rest Service Wizard .......................................................... 39  
  - Using JSON Payload or an Existing XSD File .......................................................... 40  
  - REST Service Tutorial .......................................................... 40  

TIBCO ActiveMatrix BusinessWorks™ REST Reference
Enforcing Basic Authentication ............................................................................ 80
Enforcing Basic Credential Mapping .................................................................. 83
The REST and JSON Palette in Studio ................................................................. 86
## Figures

<table>
<thead>
<tr>
<th>Figure Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST Service Wizard</td>
<td>37</td>
</tr>
<tr>
<td>Fault Handler Attached to an Inner Scope</td>
<td>71</td>
</tr>
<tr>
<td>Example of using the Catch block</td>
<td>73</td>
</tr>
</tbody>
</table>
TIBCO Documentation and Support Services

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

https://docs.tibco.com

Product-Specific Documentation

Documentation for TIBCO products is not bundled with the software. Instead, it is available on the TIBCO Documentation site. To directly access documentation for this product, double-click the following file:

TIBCO_HOME/release_notes/TIB_BW_version_docinfo.html

The following documents for this product can be found on the TIBCO Documentation site:

- Concepts
- Installation
- Getting Started
- Application Development
- Administration
- Bindings and Palettes Reference
- Samples
- Error Codes
- API Reference
- Migration

How to Contact TIBCO Support

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

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:
  http://www.tibco.com/services/support
- If you already have a valid maintenance or support contract, visit this site:
  https://support.tibco.com
  Entry to this site requires a user name and password. If you do not have a user name, you can request one.

How to Join TIBCO Community

TIBCO Community is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCO Community offers forums, blogs, and access to a variety of resources. To register, go to the following web address:

https://community.tibco.com
What is REST?

Representational State Transfer (REST) is a platform-and-language-independent architectural style used in building services for distributed systems and networked applications. REST ignores the details of component implementation and protocol syntax. It uses HTTP to make calls between the applications.

REST is primarily used to expose resources across the networks allowing applications to share data. REST APIs are used for machine-to-machine communications. REST calls (requests) and responses are sent over the HTTP protocol, hence REST requests are in the form of URLs that point to the resource(s) on the server. Required parameters are attached to the end of the URL. For example, in the resource URL http://<some-name>.com/person/1234, person is the resource and 1234 is the parameter that is passed to the URL of the resource. You can use any REST client to make REST calls.

The key features of REST architectural style are:

- **Client-server architecture**: Provides a separation of implementation details between clients and servers.
- **Stateless communication**: During a client-server communication, the server does not store any session information, making the client-server communication stateless. Every request is complete in itself and must include all the information required to complete the request.
- **Cacheability**: Provides an option to the client to cache response data and reuse it later for equivalent requests; thus partially eliminating some client-server interactions. This results in improved scalability and performance.

What is a Resource?

REST APIs operate on resources that are defined within a REST interface file such as a Swagger specification. A resource is a representation of a thing (a noun) on which the REST APIs (verbs) operate. Some examples of a resource are a user, a book, or a pet. The REST operations, POST, GET, PUT, and DELETE operate on a resource. Individual instances of a resource are identified by a unique identifier within the resource such as an ID or name.

A resource can be thought of as an entity which is expressed by a well-formed URI. In many ways it is similar to an instance of a class in the Java language. A resource has a type, one or more parameters, and some standard operations that allow you to manipulate or retrieve it from a remote location if you know its endpoint URL. The operations allowed on a resource are POST, GET, PUT, and DELETE that correspond to the CRUD operations. A resource can exist independently or it can be a part of a homogeneous collection. All the information that is relevant to a resource is contained within the resource itself.

A resource is represented by an XML object in TIBCO Business Studio™ and a JSON object in a Swagger file.

What are Operations?

Operations define the action that can be performed on the resource. REST supports POST, GET, PUT, and DELETE operations that correspond to the CRUD operations.

In TIBCO Business Studio, a process implements an operation on that resource and acts to receive, manipulate, and return resources. You pass information to an operation by attaching path or query parameters to the URL of the resource. In addition you can use header parameters to pass or receive information in the HTTP envelope containing the message body.

What is an Endpoint?

The REST API is exposed through an endpoint. It is the access point of a resource which a REST service exposes and a REST reference invokes. An endpoint uses the REST operations to provide access to the
resource. An endpoint has a name and is represented by a path within the resource. It resides at a location specified by the URL of the resource.
What is Swagger?

Swagger is an specification which is used to define a REST interface that has operations and parameters. Documents used by the REST API to send requests and receive responses are often written according to the Swagger specification.

For more details on Swagger, see http://swagger.io/.

supports the import of Swagger-compliant files that were created outside of Studio as well as allows you to generate a Swagger-compliant JSON file when you create a service from an XSD within Studio.

The Swagger UI

Using the Swagger UI you can visualize and test RESTful services defined by the Swagger specification. It specifies the format (URL, method, and representation) to describe the REST web services.

Viewing Multiple REST Services in Swagger UI

If there are multiple REST services using different HTTP Connectors, you can select the required HTTP Connector to receive responses for all the REST services using that connector. From the Swagger UI, select the connector from the options in the drop down list Select HTTP Connector.

At runtime, in the Swagger UI, the REST service lists down all the REST services that are using the selected HTTP Connector.
Parameter Support

REST operations support path, query, form, and header parameters. You can pass path and query parameters to an operation by appending them to the request URL. Header parameters are used to pass and receive information in the HTTP envelope containing the message body.

**Path** parameters can be applied only at the root level. They apply to all operations and cannot be defined per operation.

**Query** parameters can be applied both at the root level when they apply to all operations or they can be defined per operation too.

**Header** parameters are defined per operation and per direction. Request headers are separate from response headers.

**Form** parameters can be defined per operation.

**Conversion Between Swagger and XML in TIBCO Business Studio**

When you create a service using a Swagger file, TIBCO Business Studio converts the Swagger definitions into XML schema elements. You use the schema elements to configure your REST operations.

You have the option to create a REST service or reference using a Swagger file or you can create them from scratch in the Studio by creating your XML schema using the Schema Editor in Studio. When you create a REST service or reference from a Swagger file, a corresponding XSD file is automatically generated in the Schemas folder of your project. When you create a REST service or reference from scratch using their respective wizard, then a corresponding Swagger file gets generated in the Service Descriptors folder of your project.

A Swagger file is a contract that must be followed. Only the originator of the Swagger file can modify it in Studio. If Studio originated the Swagger file then Studio can modify it.

Studio maintains a link between the Swagger file and its generated .xsd file. Multiple XSD files may be linked to one Swagger file. Do not edit a Studio-generated .xsd file. Its contents will be replaced the next time the file is generated.

Not all artifacts in JSON have a direct equivalent in XML. For example, Studio handles Swagger to XML conversion of arrays differently than single elements. This sections explains how TIBCO Business Studio models the conversion of elements from Swagger to XML and vice versa.

**Basic type elements**

The following table shows the conversion of elements of basic types between XML and Swagger in TIBCO Business Studio:

<table>
<thead>
<tr>
<th>XSD type</th>
<th>Corresponding type in Swagger</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>integer</td>
</tr>
<tr>
<td>short, int, integer</td>
<td>integer</td>
</tr>
<tr>
<td>double</td>
<td>number</td>
</tr>
<tr>
<td>float</td>
<td>number</td>
</tr>
<tr>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td>XSD type</td>
<td>Corresponding type in Swagger</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>base64Binary</td>
<td>string</td>
</tr>
<tr>
<td>decimal</td>
<td>number</td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>byte</td>
<td>string</td>
</tr>
<tr>
<td>date</td>
<td>string</td>
</tr>
<tr>
<td>dateTime</td>
<td>string</td>
</tr>
<tr>
<td>binary</td>
<td>string</td>
</tr>
</tbody>
</table>

**Objects**

The following table shows how an object in JSON is converted into an XML schema element in TIBCO Business Studio. In this example, `Product` is an object that has three attributes called 'product_id', 'description' and 'display_name' all of which are of type string.

Since the object has multiple attributes, it is a complex type element in XSD. The `minOccurs="0"` indicates that specifying a value for the attribute is optional.

**Arrays**

An array is a collection of identically typed elements. The type can be primitive or complex. For the most part, when Studio converts from JSON to XSD, you can see a one-to-one correspondence for the objects in Swagger and elements in the corresponding XSD file. The only exception lies in the handling of arrays.

The word "Array" is a key word in Studio. Do not use the "array" suffix in an XSD element name.

**Swagger array representation in Studio**
When TIBCO Business Studio encounters an array in the Swagger file, while generating a schema for it, TIBCO Business Studio models the array by generating two separate but related elements in the .xsd file for each array:

- a wrapper element (with an "Array" suffix) that acts as a definition for a container that holds the array elements. In addition to other attributes, this wrapper element contains the type of the element that the array contains. The wrapper element is a Studio-generated artifact solely to comply with the XML requirement of having a container for a collection. It does not exist in the .json file. The array element is created with a boundary of 0..* (0 indicates that it is optional and * indicates that the array is unbounded).
- a definition of the element itself.

Do not edit a Studio-generated .xsd file. Its contents will be replaced the next time the file is generated.

The example below shows the definition of an array called Products in Swagger and its corresponding XSD.

<table>
<thead>
<tr>
<th>Array in JSON...</th>
<th>represented in XSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Products&quot;: {</td>
<td>&lt;xs:element name=&quot;Products&quot; type=&quot;tns:Products&quot;/&gt;</td>
</tr>
<tr>
<td>&quot;type&quot;: &quot;array&quot;,</td>
<td>&lt;xs:complexType name=&quot;Products&quot;&gt;</td>
</tr>
<tr>
<td>&quot;items&quot;: {</td>
<td><a href="">xs:sequence</a></td>
</tr>
<tr>
<td>&quot;$ref&quot;: &quot;Product&quot;</td>
<td>&lt;xs:element maxOccurs=&quot;unbounded&quot; minOccurs=&quot;0&quot; name=&quot;Product&quot; type=&quot;tns:Product&quot;/&gt;</td>
</tr>
<tr>
<td>}</td>
<td>&lt;/xs:sequence&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:complexType&gt;</td>
</tr>
</tbody>
</table>

In the example above, Products is an array in JSON (denoted by "type": "array") that contains multiple Product objects (denoted by "items": { "$ref": Product}). The object, Product, itself is defined in another location in the Swagger file. (see the "Objects" section above for the definition of Product).

The following shows how the Products array defined above is used as a path parameter:

<table>
<thead>
<tr>
<th>The following in JSON schema...</th>
<th>...gets converted to the following in XSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;schema&quot;: {</td>
<td>&lt;xs:element name=&quot;ProductArray&quot; type=&quot;tns:ProductArray&quot;/&gt;</td>
</tr>
<tr>
<td>&quot;type&quot;: &quot;array&quot;,</td>
<td>&lt;xs:complexType name=&quot;ProductArray&quot;&gt;</td>
</tr>
<tr>
<td>&quot;items&quot;: {</td>
<td><a href="">xs:sequence</a></td>
</tr>
<tr>
<td>&quot;$ref&quot;: &quot;Product&quot;</td>
<td>&lt;xs:element maxOccurs=&quot;unbounded&quot; minOccurs=&quot;0&quot; name=&quot;Product&quot; type=&quot;tns:Product&quot;/&gt;</td>
</tr>
<tr>
<td>}</td>
<td>&lt;/xs:sequence&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xs:complexType&gt;</td>
</tr>
</tbody>
</table>

The example above appear as follows in the Schemas folder of Studio:
Anonymous Arrays

Since XML is a strongly-typed language, all elements - arrays and single elements alike, are named and have a type in XML. In JSON however, arrays can either be structured or anonymous. A structured array is type-based where a type defines the basic construct and also its elements. An anonymous array is an unnamed construct containing a group of homogenous objects. Neither the construct nor the elements contained in it have a type. Anonymous arrays simply contain blocks of data that repeat. The concept of anonymous arrays is used extensively in JSON, but does not exist in XML. In JSON, a parameter may be of type string, but if you add “type” : “array” to the definition, it becomes a collection of strings.

The following is an example of a JSON payload and its equivalent in XSD in Studio. The wizard prompts you to enter a file name when generating XSD from a JSON payload. The file name entered was "ClassicNovels" in this example.
<table>
<thead>
<tr>
<th>Anonymous array in JSON...</th>
<th>...gets converted to the following in XSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{</td>
<td><code>&lt;complexType</code></td>
</tr>
<tr>
<td>&quot;id&quot;: 1,</td>
<td>name=&quot;ClassicNovelsElementType&quot;&gt;</td>
</tr>
<tr>
<td>&quot;name&quot;: &quot;Great</td>
<td>&lt;sequence&gt;</td>
</tr>
<tr>
<td>Expectations&quot;,</td>
<td>&lt;element maxOccurs=&quot;1&quot;</td>
</tr>
<tr>
<td>&quot;author&quot;: &quot;Charles</td>
<td>minOccurs=&quot;0&quot; name=&quot;id&quot; type=&quot;integer&quot;/&gt;</td>
</tr>
<tr>
<td>Dickens&quot;,</td>
<td>&lt;element maxOccurs=&quot;1&quot;</td>
</tr>
<tr>
<td>&quot;isbn&quot;: &quot;13: 978-0141439563&quot;,</td>
<td>minOccurs=&quot;0&quot; name=&quot;name&quot;</td>
</tr>
<tr>
<td>&quot;genre&quot;: &quot;Classic&quot;</td>
<td>type=&quot;string&quot;/&gt;</td>
</tr>
<tr>
<td>},</td>
<td>&lt;/element&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/sequence&gt;</td>
</tr>
<tr>
<td>`{</td>
<td>&lt;/complexType&gt;</td>
</tr>
<tr>
<td>&quot;id&quot;: 2,</td>
<td>name=&quot;ClassicNovelsElement&quot; type=&quot;tns:ClassicNovelsElementType&quot;/&gt;</td>
</tr>
<tr>
<td>&quot;name&quot;: &quot;Jane</td>
<td>complexType</td>
</tr>
<tr>
<td>Austen&quot;,</td>
<td>name=&quot;ClassicNovelsElementArrayType&quot;&gt;</td>
</tr>
<tr>
<td>&quot;author&quot;: &quot;Emma&quot;,</td>
<td>&lt;sequence&gt;</td>
</tr>
<tr>
<td>&quot;isbn&quot;: &quot;13: 978-1493663644&quot;,</td>
<td>&lt;element maxOccurs=&quot;unbounded&quot;</td>
</tr>
<tr>
<td>&quot;genre&quot;: &quot;Romance&quot;</td>
<td>minOccurs=&quot;0&quot; ref=&quot;tns:ClassicNovelsElement&quot;/&gt;</td>
</tr>
<tr>
<td>},</td>
<td>&lt;/element&gt;</td>
</tr>
<tr>
<td>`{</td>
<td>&lt;/complexType&gt;</td>
</tr>
<tr>
<td>&quot;id&quot;: 3,</td>
<td>name=&quot;ClassicNovelsElementArray&quot; type=&quot;tns:ClassicNovelsElementArrayType&quot;/&gt;</td>
</tr>
<tr>
<td>&quot;name&quot;: &quot;Jude the</td>
<td>element name=&quot;ClassicNovelsElement&quot;</td>
</tr>
<tr>
<td>Obscure&quot;,</td>
<td>type=&quot;tns:ClassicNovelsElementType&quot;/&gt;</td>
</tr>
<tr>
<td>&quot;author&quot;: &quot;Thomas</td>
<td>complexType</td>
</tr>
<tr>
<td>Hardy&quot;,</td>
<td>name=&quot;ClassicNovelsElementArrayType&quot;&gt;</td>
</tr>
<tr>
<td>&quot;isbn&quot;: &quot;13: 978-0140435382&quot;,</td>
<td>&lt;sequence&gt;</td>
</tr>
<tr>
<td>&quot;genre&quot;: &quot;Tragedy&quot;</td>
<td>&lt;element maxOccurs=&quot;unbounded&quot;</td>
</tr>
<tr>
<td>}</td>
<td>minOccurs=&quot;0&quot; ref=&quot;tns:ClassicNovelsElement&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/element&gt;</td>
</tr>
</tbody>
</table>

The above example appears as follows in the Schemas folder in Project Explorer:
Forms

TIBCO Business Studio supports the use of form parameters as the media type in REST requests. This is the only media type that can be used to transmit binary data and files. Form parameters can be created at both the root level when they apply to all operations or they can be defined per operation.

An operation in a REST API has one of the following encoding:

- **Tag/Value** (application/x-www-form-urlencoded) - used when you use form data which is of a primitive data type. You cannot send or receive binary data or files using this encoding.

- **Multipart** (application/form-data) - is superset of urlencoded encoding. Besides primitive data types, multipart encoding also supports binary and file data types. When the data type of a form parameter is either binary or file, two elements get created in Studio:
  - **name** - used to store the name of the file
  - **content** - used to store the actual data within the file

The following illustrates how form parameters are represented in JSON and XSD:
<table>
<thead>
<tr>
<th>Data types</th>
<th>JSON</th>
<th>XSD</th>
</tr>
</thead>
</table>
| primitive data types | `{  
  "/person" : {  
    "post" : {  
      "description" : ",  
      "operationId" : "post-person",  
      "consumes" :  
      [ "application/x-www-form-urlencoded" ],  
      "produces" :  
      [ "application/json" ],  
      "parameters" : [ 
        {  
          "name" : "name",  
          "in" : "formData",  
          "description" : "",  
          "type" : "string",  
          "required" : false,  
          "allowMultiple" : false  
        }  
      ]  
    }  
  }  
}` | `<xs:element name="personPostForm">  
  <xs:complexType>  
  <xs:sequence>  
    <xs:element minOccurs="1" maxOccurs="1" name="name" type="xs:string"/>  
  </xs:sequence>  
  </xs:complexType>  
</xs:element>` |
| binary or file type | `{  
  "/person" : {  
    "post" : {  
      "description" : ",  
      "operationId" : "post-person",  
      "consumes" :  
      [ "multipart/form-data" ],  
      "produces" :  
      [ "application/json" ],  
      "parameters" : [ 
        {  
          "name" : "name",  
          "in" : "formData",  
          "description" : "",  
          "type" : "file",  
          "required" : false,  
          "allowMultiple" : false  
        }  
      ]  
    }  
  }  
}` | `<xs:element name="personPostForm">  
  <xs:complexType>  
  <xs:sequence>  
    <xs:element minOccurs="0" maxOccurs="1" name="photo" type="xs:string"/>  
  </xs:sequence>  
  </xs:complexType>  
</xs:element>` |

The above examples appear as follows in the Studio:

**Primitive type:**

```json
{
  "/person" : {
    "post" : {
      "description" : ",
      "operationId" : "post-person",
      "consumes" : [
        "application/x-www-form-urlencoded"
      ],
      "produces" : [
        "application/json"
      ],
      "parameters" : [
        {
          "name" : "name",
          "in" : "formData",
          "description" : "",
          "type" : "string",
          "required" : true,
          "allowMultiple" : false
        }
      ]
    }
  }
}
```
Working with Path and Query Parameters

Path and query parameters are supported in TIBCO ActiveMatrix Businessworks REST APIs. Path parameters can be applied only at the root level when they apply to all operations. They cannot be defined for individual operations only. Query parameters can be applied at root as well as at individual operation level.

Define parameters in the resource service path by enclosing each parameter in {} brackets. For example, to define the path parameter isbn for a book resource, specify the resource path as follows:

/book/{isbn}

In this example, the client would invoke this service using the URL http://<host>:<port>/book/<isbn>.

Insert a question mark (?) after a parameter to add query parameters to a resource. In the following example, isbn is defined as a query parameter, instead of the path parameter, in the resource service path:

/book?{isbn}

Insert (&) to use it as a delimiter when defining multiple query parameters. In the following example, the query parameters isbn and authorName are defined in the resource service path:

/book?{isbn}&{authorName}
All the parameters defined in the resource service path are made available to the user as an Input for every operation. See the following image for the /book/{isbn} example to see the Input for the GET operation.

Path parameter names and query parameter names must be unique in a resource service path. For example, the following path and query parameters is not supported:

```
/books/{isbn}?{isbn}
```

If the Request Format for a REST Binding is Form, the parameter name must be different from all element names in the referenced schema. For example if a Books schema contains the element `isbn`, `isbn` can not be used as query or path parameter name.

Do not use the word "fields" as the name of a query parameter. The word "fields" is a reserved keyword for internal use by BusinessWorks. For example, to extract the author, isbn, and price fields from an instance of a book object with name Emma:

- use: `/books/Emma?author,isbn,price`
- not: `/books/Emma?fields=author,isbn,price`

### Adding or Editing Path Parameters

You can create or modify path parameters for REST services that were created in the Studio using an XSD. For REST services that were created using a Swagger file that was imported from an external source, you can only view the parameters that its operations support. You cannot create or modify the parameters for such services.

To create a path parameter, do the following:

**Procedure**

1. Click **Components** under the **Module Descriptors** to open the **Component Configurations** page.
2. Expand the **Component<application_name>** node.
3. Double-click the process name (with the green chevron next to it) to open its properties.
4. Click the **Bindings** tab.
5. Append the path parameter to the **Resource Service Path**. For example, to define the path parameter `isbn` for a book resource, specify the resource path as `/book/{isbn}`.
6. Click the green check mark button at the end of the text box to save your edit.

The newly created path parameter appears in the Path Parameters table. You can change the name of the parameter by clicking on it. You can change its type by clicking on it and selecting a new type from its drop-down menu. Be aware that these buttons will be disabled if you are viewing operations for an API that was created outside the Studio and imported into the Studio.
Adding or Editing Query Parameters

You can create or modify query parameters for REST services that were created in the Studio using an XSD. For REST services that were created using a Swagger file that was imported from an external source you can only view the parameters that its operation supports. You cannot create or modify the parameters in such services.

To create a query parameter, do the following:

Procedure

1. Click **Components** under the **Module Descriptors** to open the **Component Configurations** page.
2. Expand the **Component<application_name>** node.
3. Double-click the process name (with the green chevron next to it) to open its properties in the Properties view.
4. Click the **Bindings** tab.
5. Click an operation name in the **Operations** section.
6. Click the **Request** tab in the **Operation Details** section.
7. Click the green icon (green box) to add a query parameter. To edit the parameter name click on the newly created parameter’s default name and type in a new name. Be aware that these buttons will be disabled if you are viewing operations in an API that was created outside the Studio and imported into the Studio.
8. You can also edit the existing query parameter to make it required or optional by clicking in the cell that corresponds to the parameter in the **Required** column. The value toggles from Yes to No or vice versa.

Working with Arrays

Swagger provides an interface description that could return JSON objects. An operation may return a single object, or if the "type": "array" attribute is added to the configuration then it returns an array of that object type. TIBCO Business Studio supports sending and receiving arrays in REST requests and responses.

In TIBCO Business Studio, you can create a REST API by starting with a Swagger-compliant JSON file or you can start create the API from scratch using the wizards in the Studio. If you use JSON as your starting point, TIBCO Business Studio generates an XSD file when the API gets created. When generating the XSD for an array, Studio creates a wrapper element with an "Array" suffix and another single element containing the definition for the element type contained in the array. Refer to the Arrays section in the topic Conversion Between JSON and XML in TIBCO Business Studio™ for a detailed explanation of how the TIBCO Business Studio™ handles arrays.

Do not edit the Studio-generated .xsd file.

The word "Array" is a key word in Studio. Do not use the "array" suffix in an XSD element name.

Configuring an array in a REST binding in Studio

For projects that were created with a Swagger file that was imported from an external source into the Studio, you can only view and use the elements. You cannot modify them.

A JSON file is like a contract that must not be broken. Since it is an imported file, its contents cannot be modified and must be followed exactly.

An array appears as follows:
To configure a REST request to get an array, do the following:

1. In Project Explorer, under Module Descriptors, double-click Components to open its property page.
2. Click on the resource to open its properties view.
3. Click the Bindings tab to open it.
4. In the Operations Details box, click the Response tab.
5. Select the JSON checkbox in the Format options.
6. Select Anonymous Array from the JSON Response Style drop-down menu.
7. Select the array element type in the Resource Schema dropdown menu.
8. Save your project.
9. Verify that the array is available for use in the Input tab for your postOut activity.
Working with Form Parameters

TIBCO Business Studio supports the use of form parameters as the media type in REST requests. This is the only media type that can be used to transmit files. Refer to the Forms section in the Conversion Between JSON and XML in TIBCO Business Studio section for details on how Form parameters are represented in JSON and XSD.

Form parameters can be created at the operation level where you can define different form parameters for different operations.

An operation can have one of the following encodings. Both encodings have Tag/Value:

- **Tag/Value (application/x-www-form-urlencoded)** - Select this encoding for an operation, if you want to use form parameters of primitive data type of String, Integer, or Boolean in your operation. You cannot use this encoding to transmit files.

- **Multipart (application/form-data)** - Select this encoding for an operation, if you want to use form parameters of type String, Integer, Boolean, File/Binary, or File/Text in your operation. You can send or receive both text and binary files.

When you transmit a file using a form parameter, two elements get created in Studio:

- **name** - used to store the name of the file
- **content** - used to store the actual data within the file

Creating Form Parameters

You can create form parameters in POST and PUT operations only. You can create or modify parameters for REST services that were created ground up in the Studio without a Swagger file. For REST services that were created using a Swagger file that was imported into Studio from an external source you can only view the parameters. You cannot create or modify the parameters in such services.

To create a form parameter, do the following:

**Procedure**

1. Click **Components** under the **Module Descriptors** of your project to open the **Component Configurations** page.
2. Expand the **Component<application_name>** node.
3. Double-click the process name (with the green chevron next to it) to open its properties in the Properties view.
4. Click the **Bindings** tab.
5. Click the **post** operation under **Operations** in the left tree.
6. Click the **Request** tab.
7. Select an encoding for the operation from the **Request** drop-down menu. Select either **Form Data - Tag/Value (application/x-www-form-urlencoded)** or **Form Data - Multipart (application/form-data)**. The Form Parameters table is displayed.
8. Click the blue icon ( guardar) on the right of the Form Parameters table to create a new form parameter.
9. Click the name and edit it. To change the type, click the type in the Type column and select a type from the drop-down list. Similarly you can specify if this parameter is required or not by clicking in the Required column.
Working with Header Parameters

Header parameters are used to define user-defined custom HTTP headers for a request, for example, the APIKey could be a HTTP Header parameter. Unlike query and path parameters, header parameters do not appear in the URL of the request, but if they exist in your API specification, they get passed into the request but are not visible to the user.

Header parameters can be defined per operation which means that each operation in the API can have its own header parameters.

If you created your REST API from an imported Swagger file, the parameters will not be editable in TIBCO Business Studio. You cannot modify existing parameters or create new ones. You can only view the parameters. Any custom header parameters you create, will be visible in the Data Source tab of the TIBCO Business Studio as follows:

You can view or add header parameters in the TIBCO Business Studio if the Swagger file for your project was auto-generated in the Studio by implementing a REST service from XSD. To add a custom header parameter, follow these steps:

1. Click Components under the Module Descriptors to open the Component Configurations page.
2. Double-click the process name (with the green chevron next to it) to open its properties in the Properties view.
3. Click the Bindings tab.
4. Click an operation name in the Operations section.
5. Click the Request tab in the Operation Details section.
6. Click the red icon to add a header parameter. To edit the parameter name click on the newly created parameter’s default name and type in a new name. Be aware that these buttons will be disabled if you are viewing operations for an API that was created outside the Studio and imported into the Studio.
7. You can also edit an existing header parameter to make it required or optional by clicking in the cell that corresponds to the parameter in the **Required** column. The value toggles from Yes to No or vice versa.

8. To add a header parameter to the operation response, do so in the **Response** tab.
About REST Services and References

A REST service or a reference is created from a process within a project. The content of the process resides in an XSD file in the Schemas folder of the project.

The Schemas folder and the Service Descriptors folder in a project are tightly coupled. If you want to create a service using a Swagger file, you must first import that Swagger file into the Service Descriptors folder of your project. While creating the project, TIBCO Business Studio automatically generates a schema (.xsd) for the definitions in your Swagger file in the Schemas folder of your project.

Service Descriptors folder

The Service Descriptors folder in a project stores the interface description files, such as the .json Swagger file specific to the activator process. These files describe the service and the list of operations and parameters that the service supports. The Service Descriptors folder provides a good way to view the structure of the API. You can use these files to create a service or create a reference to invoke the service.

Schemas folder

The Schemas folder is used to store the XSD definitions that were either generated by Studio or imported into the Studio from an external source.

Approaches to Creating a Service or Reference

Use one of the following approaches to create a REST service or reference:

- Using a Swagger file
- Starting from ground up without a Swagger file
Supported Message Formats

ActiveMatrix BusinessWorks supports the following message formats: JSON, XML.
Restrictions

There are certain rules to follow when working with JSON and XML Schema.

Restrictions on JSON

JSON files used to create REST services and references have some restrictions.

- Arrays must have homogeneous content.
- Arrays cannot directly contain arrays ([ [ ... ], [ ... ] ])

Not currently supported

Binary content in JSON as a special case

Restrictions on XML Schema

This topic lists the restrictions on XML Schema.

Schema Restrictions

- No wildcards or attribute wildcards. For example, any element and any attribute is not supported.
- No foreign-namespace elements or attributes (single namespace).
- Complex types may not contain both an attribute and a child element with the same local name.
- Complex types may not be of the pattern "simple type plus attributes".
- Complex types may not contain mixed content.
- Attributes that are not part of the default(empty) namespace, cannot be used for Complex Elements.
- The 'choice' and 'sequence' compositors may not have maxOccurs > 1 (same as the restriction on 'all' in the schema specification).
- Substitution groups are not allowed.
- Element of simple type with an attribute is not allowed.
- The elementFormDefault can only be qualified for schemas used by REST binding and JSON activities.

Restrictions on JSON

- Arrays must have homogeneous content.
- Arrays cannot directly contain arrays ([ [ ... ], [ ... ] ])

Not currently supported

- Binary content in JSON as a special case
Using Swagger 1.2 Files

You can use Swagger 1.2 API declaration files.
The Swagger 1.2 files were not exposed in previous versions of BusinessWorks. To consume the interfaces created using Swagger 1.2 in the current version, import the projects, and click the REST module under the Processes folder in the Project Explorer and select Refactor -> Expose REST Resource to extract the hidden resource files. These Swagger 1.2 files appear in the Service Descriptors folder and can be used in the same manner as the Swagger 2.0 files.
REST Schema Utilities

TIBCO Business Studio provides utilities that allow you to convert objects from JSON to XSD elements.

Creating a New XML Schema File

TIBCO Business Studio allows you to create a new XML schema file in your project.

Prerequisites

A project must exist in the Project Explorer.

Procedure

1. Right-click the Schemas folder in your project in the Project Explorer.
3. Enter a file name and click Finish.
The file opens in the XML Schema Editor.
4. Right-click anywhere in this editor and click Add Element to add a new element.

Creating XML Schema From a JSON Payload

TIBCO Business Studio provides a utility that allows you to generate an XML schema from a JSON payload.

To generate an XML schema from a JSON payload, do the following:

Procedure

1. Right-click on the Schemas folder in your project and select New > XML Schema File from JSON Payload.
2. Enter a name for the schema file and paste the JSON payload into the JSON Sample text box and click Next.
You should see a message saying that the JSON payload parsed successfully.
3. Click Finish.
A .xsd file with the name that you provided gets created under the Schemas folder. You can view its contents by opening it in the XML Schema Editor (right-click schema file and select Open With > XML Schema Editor).
4. Save the project.

**Generating an XML Schema from a Swagger File**

TIBCO Business Studio automatically generates a Swagger file when you create a REST service from an XML schema. It also allows you to generate an XML schema from a Swagger 1.0 or 2.0 file using the menu options without creating a service.

This option is enabled only when the JSON file is a Swagger 1.0 file or Swagger 2.0 file and contains Swagger Definition. For Swagger files that do not contain Swagger Definition, the Generate XSD Schema menu option is disabled.

**Prerequisites**

The Swagger 2.0 file should exist in the Service Descriptors folder of the project. Be sure to import the Swagger file into the Service Descriptors folder before you follow the steps to create its XML schema.

**Procedure**

1. Right-click the JSON file in the Service Descriptors folder in your project and select Refactor > Generate XSD Schema. A .json file gets created in the Service Descriptors folder of your project.

2. Save the project.
   - To find out which XML schema is related to the Swagger file, right-click the Swagger file and select Refactor -> Open XSD Schema.
   - If you have multiple Swagger files all of which contain a definition for the same object, the definition for the object in all the Swagger files must be identical.
   - If you have multiple Swagger files with one file (a master file) containing a super set of definitions contained in the other files, generate an XSD file from the master Swagger file that contains the super set, and create links to the other files in the master Swagger file. If you create a link to the super set file in one of the subset files and then create an XSD from the subset file, then the XSD will contain only those elements that are common to both files. It will not contain elements for definitions that exist only in the super set file.
REST Service

The REST service is a server process. When running, it can be invoked by a REST reference.

REST Service Binding

REST Binding provides external connectivity for REST over HTTP. You can specify custom HTTP headers and parameters using REST binding. It supports GET, PUT, POST, and DELETE HTTP methods. It also supports JSON, XML, and plain text message types.

Binding

This section has the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
<td>The name of the resource.</td>
</tr>
<tr>
<td>Resource Service Path</td>
<td>Specify the path of the Service Resource.</td>
</tr>
<tr>
<td></td>
<td>If an application contains multiple REST bindings, ensure that the location of the path parameters is unique for each REST binding.</td>
</tr>
<tr>
<td></td>
<td>For example, one REST binding is using the paths /book/{isbn} and another REST binding is using the path /book/{authorid}. Since {isbn} and {authorid} are defined at the same location in the URI, one of these services will not function correctly.</td>
</tr>
<tr>
<td>HTTP Connector Name</td>
<td>The name of the HTTP Connector.</td>
</tr>
<tr>
<td></td>
<td>Click on the HTTP Connector Name field to display details about the HTTP Connector resource.</td>
</tr>
<tr>
<td>Reply Client Format</td>
<td>The type of reply message format.</td>
</tr>
<tr>
<td></td>
<td>The two available reply message format options are: JSON and XML.</td>
</tr>
</tbody>
</table>

Imported projects display the Authenticate check box under the Binding section if the check box was selected in a previous version TIBCO ActiveMatrix BusinessWorks 6.x. Authentication also remains enabled on the REST Service Binding if you do not deselect the check box. Deselecting the Authenticate check box causes a warning message to display prompting you for confirmation. Clicking OK causes the Authenticate check box to no longer display and removes authentication from REST Service Binding. Click Cancel to retain the Authenticate check box, and to continue enforcing authentication on the binding.

To enforce authentication on the REST Service Binding, associate the binding with a Basic Authentication policy. For additional details, refer to the sections "Enforcing Basic Authentication" in the TIBCO ActiveMatrix BusinessWorks Application Development guide.

Operations

This section shows the following details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the HTTP method used, for example, GET, PUT, POST, and DELETE.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>Nickname</td>
<td>The specified name of the service, for example, getBooks.</td>
</tr>
</tbody>
</table>

**Operation Details**

This section shows the following details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>This tab has the following available options:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Summary</strong>: The summary of the REST resource</td>
</tr>
<tr>
<td></td>
<td>• <strong>HTTP Method</strong>: Displays the HTTP Method specified in the Operations section. These are the available HTTP methods:</td>
</tr>
<tr>
<td></td>
<td>- POST</td>
</tr>
<tr>
<td></td>
<td>- GET</td>
</tr>
<tr>
<td></td>
<td>- PUT</td>
</tr>
<tr>
<td></td>
<td>- DELETE</td>
</tr>
<tr>
<td>Request</td>
<td>Specifies the resource created, using the POST or PUT method. Parameters supported are body, form, query, and header parameters.</td>
</tr>
<tr>
<td>Response</td>
<td>This tab has the following available options:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Use HTTP Headers</strong>: Selecting this check box includes the response headers element. Response headers are not commonly used, so select this checkbox only when you need to include response headers.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Use Custom Status Line</strong>: You can specify a custom status line (status code and reason phrase) to the outgoing message. The codes used must be defined in the configuration under the Response Status tab.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Response with Status Code Only</strong>: The operation returns a status code as response, when this check box is selected. Message body is not required. For example, using a POST operation returns a 201 status code which means “Created” and responds with the resource URL.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Resource Schema</strong>: Displays the schema selected. This option is not available when the Use Custom Status Line and Response with Status Code Only check boxes are selected. These are the available options:</td>
</tr>
<tr>
<td></td>
<td>- String</td>
</tr>
<tr>
<td></td>
<td>- Integer</td>
</tr>
<tr>
<td></td>
<td>- Boolean</td>
</tr>
<tr>
<td></td>
<td>- <strong>XSD element</strong>: Selecting this option to either select the XSD schema element available under the Schemas folder of your project or a create new XML schema resource. Click Create New Schema to a create new XML schema resource using the Simplified Schema Editor wizard.</td>
</tr>
</tbody>
</table>
Response Status | Specifies the response code for the operation and the code message. For example, code 201 means Created or code 503 means Service Unavailable. You can also add your own custom code and reason phrase.

Path Parameters

This section shows the following details.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter name of the operation used</td>
<td>The parameter type. It can be any one of the following:</td>
</tr>
<tr>
<td></td>
<td>● String</td>
</tr>
<tr>
<td></td>
<td>● Integer</td>
</tr>
<tr>
<td></td>
<td>● Boolean</td>
</tr>
<tr>
<td></td>
<td>● Long</td>
</tr>
<tr>
<td></td>
<td>● Float</td>
</tr>
<tr>
<td></td>
<td>● Double</td>
</tr>
<tr>
<td></td>
<td>● Byte</td>
</tr>
<tr>
<td></td>
<td>● Binary</td>
</tr>
<tr>
<td></td>
<td>● Date</td>
</tr>
<tr>
<td></td>
<td>● DateTime</td>
</tr>
<tr>
<td></td>
<td>● Password</td>
</tr>
</tbody>
</table>

Policies

To associate a new or existing policy with the REST binding, click Policies node under the RestService tree, then click the Add Policy button to create a new policy. To edit an existing policy details, click the policy name and edit any of the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy Name</td>
<td>The name of the policy.</td>
</tr>
<tr>
<td>Policy Type</td>
<td>The type of policy associated with the binding. The REST binding can support the Basic Authentication policy.</td>
</tr>
<tr>
<td>Description</td>
<td>A description of the policy.</td>
</tr>
</tbody>
</table>
Creating a REST Service

A service is created from a process. You expose the process and describe the content that is sent and received by the process in an XSD. The XSD defines the data that you send and receive. The process is the actual implementation of what you do with the data.

The key abstraction of information in REST is a resource. REST ignores the details of component implementation and protocol details. ActiveMatrix BusinessWorks currently allows the following HTTP methods to be performed on resources: GET, PUT, DELETE, and POST. Both XML and JSON are supported as data serialization formats along with support for definition of custom status codes, path(URL) parameters, key-value parameters, query parameters, form parameters, and custom HTTP headers.

You can create a REST service in TIBCO Business Studio in one of the following ways:

- Using a Swagger file
- Without a Swagger file by creating a process and adding a REST resource to it using the REST Service wizard

Using Swagger to Create a REST Service

You can drag and drop a path from the Swagger file on to the left boundary of the Process Editor to create a REST service or drag it to the center and select Create Service from the resulting menu.

When you create a REST service, make sure to edit the Default Host field in the HTTP Connection Resource to reflect the actual host name. By default, the Default Host field is set to localhost using $BW.HOST.NAME$ module property.

When you use Swagger to create a REST service, the fields in the Bindings tab of the service properties display as read-only. Swagger is like a contract that must be followed exactly, so the service you create with the Swagger file cannot be modified.

To create a REST service from a Swagger file, do the following:

Procedure

1. Create an empty project if you do not already have one. Refer to the topic "Developing a Basic Process" in the Application Development Guide.
2. Import the Swagger JSON file into the Service Descriptors folder of your project by dragging and dropping it from the File Explorer view.
3. Expand the .json node that you just created under the Service Descriptors folder to view the available paths.
4. Drag and drop a path from the Service Descriptors folder to the left side of the process editor to create a service or to the right side of the process editor to create a reference. TIBCO Business Studio automatically generates a corresponding XSD schema for the Swagger file in the Schemas folder.

Using the Wizard to Create a REST Service

A REST service provider exposes the resources in a process definition that can be invoked by clients using one of the following operations- POST, GET, PUT, and DELETE.

Prerequisites

If a schema definition does not exist, create (or import) a schema definition in the process to which you want to add the REST service. To import an existing XSD file, drag and drop the .xsd file from the File Explorer to the Schemas folder of your project.
To create a new schema file in Studio, right-click the Schemas folder in Project Explorer, and select **New > XML Schema File**. This opens the Schema Editor in Studio. Right-click in the respective box in the Schema Editor to add a directive, element, type, attribute, or group.

When you create a REST service, make sure to edit the **Default Host** field in the HTTP Connection Resource to reflect the actual host name. By default, the Default Host field is set to localhost.

**Procedure**

1. In the Project Explorer, select the process to which you want to add the REST service. There are multiple ways to invoke the wizard to create a REST service.
   - From the main menu, select **File > New > BusinessWorks Resources > BusinessWorks REST Resource**.
   - Right-click the menu, select **New > BusinessWorks REST Resource**.
   - Click on **Create REST Service** in the process editor area. (Note that REST services can only be created in stateless BusinessWorks processes.)

   ![Create REST Service]

   For more information, see the **REST Binding** in the Binding and Palettes Reference guide.

2. In the REST Service Wizard window, configure the REST service implementation by specifying the values for Resource Service Path, Type of Resource, Operations, and Implementation Data.
• **Resource Name** a name for the new REST service
• **Summary** about the new REST service.
• **Resource Service Path**: Specifies the URI that is used to access the REST service.
• **Resource Definition**: Select a resource schema for the REST service, if needed.
• **Operations**: By default, the POST operation is selected. Select or deselect the operations as needed.
• **Implementation Data**: Choose between structured and opaque implementation data.

3. Optionally, click **Next** to configure the selected operations individually to specify the nickname for the operation (default nickname is of the format `<operation><resource_name>`), summary, and the request and response elements and their data types.

4. Click **Finish**.
   The wizard adds the REST service and the selected operations, and also creates a process definition with the multiple operations. It generates a `.json` file in the **Service Descriptors** folder of your project when it creates the service.

   The REST service always implements the constructor operator.
5. Add activities to the process and configure them appropriately. For example, update the POST process to add a Log activity to log the requests and connect the postOut activity to Log activity.

6. Configure the input and output properties for the activities. For example, select postOut activity and then select Properties > Input. Expand the data tree in the Data Source tab and map the post element from the left to the post Response element on the right to echo the element. Similarly, for Log activity, map the post element on the left to the ActivityInput message element on the right.

7. You can optionally add an operation to the service using the Create REST Operation wizard. To open this wizard, click on the down arrow and click Create REST Operation to open the wizard.

8. Save your changes.

**Result**

The REST service is built and can be tested using the built-in tester Swagger UI. For more information on Swagger UI, see Testing the REST Service in the Getting Started guide.
Rest Service Wizard

Rest Service Wizard is used to create a new Rest Resource or add Rest Services to an existing resource in TIBCO Business Studio.

The Rest Service Wizard has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Name</td>
<td>The name of the REST Resource.</td>
</tr>
<tr>
<td>Summary</td>
<td>The summary or description of the REST Resource.</td>
</tr>
<tr>
<td>Resource Service Path</td>
<td>The relative path for this REST Service.</td>
</tr>
<tr>
<td>Resource Definition</td>
<td>The XSD schema element to be used for creating the REST resource. You can also use this to create the input and output of each operation defined. You can override this on the next screen if required, for each operation.</td>
</tr>
<tr>
<td>Operations</td>
<td>These are the HTTP methods implemented by this REST Service. Currently only POST, GET, PUT and DELETE methods are supported for users to implement.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Implementation Data</th>
<th>The implementation data field can be <strong>Structured</strong> or <strong>Opaque</strong>.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- <strong>Structured</strong>: The XSD element structure is preserved for the input and output of every operation. You need not manually parse the payload to generate the actual element to be used in the process.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Opaque</strong>: Use this mode to apply the pass through mechanism. A user gets a messageBody element in the input or output of every operation and needs to use either <strong>Parse</strong> activities for JSON or XML to get a Structured output for the payload.</td>
</tr>
</tbody>
</table>

To add additional services to a process, click the **Create A Rest Service** icon on the top left of the process canvas.

Re-creating a component containing a REST Binding after deleting the component is not supported. To add the REST Binding, in the process editor, right click on the service without a binding. Go to **Components > [componentware] > Create REST Binding**. A binding is created for the service. The binding will have to be re-configured as the previous configurations will be lost.

### Using JSON Payload or an Existing XSD File

If you would like to use an existing JSON Payload or XSD file to create a service, you can do so by importing them into the TIBCO Business Studio.

To use an existing XSD file, drag and drop the XSD file from the File Explorer into the Schemas folder of your project. Then use the REST Service Wizard to create a service by following the instructions in **Using the Wizard to Create a REST Service** section.

To use a JSON payload, create an XSD schema with the JSON payload. See **Creating XML Schema From a JSON Payload** for details on it. Then use the REST Service Wizard to create a service by following the instructions in **Using the Wizard to Create a REST Service** section.

do the following:

### REST Service Tutorial

The REST Bookstore sample lets you explore the REST tooling in TIBCO Business Studio™. You can import this sample into TIBCO Business Studio™ through **File Explorer** and examine the project and the solution implemented by it.

The processes in the sample implement different aspects of a bookstore, such as adding books, deleting a book, and getting a list of books or a single book by ISBN. For more information about the sample, see the topic called "Using REST to Manage Books for a Bookstore" in the **Samples** guide. This tutorial walks you through the steps to build an additional REST service for the sample and test it in the debugger. You can use the Swagger UI to invoke the operations for the REST resource.

### Prerequisites

Before starting the tutorial, you must have PostgreSQL installed on your machine, with the required database and tables created. See the topic called **Installing PostgreSQL** for information.

Install the latest version of Google Chrome.
Installing PostgreSQL

This topic explains how to install the PostgreSQL database and create the database and tables required for the Bookstore tutorial.

Procedure

   Note the superuser password that you create as part of the installation process.

   If installing on Windows, do not install or run as Administrator.

2. Open a terminal window and navigate to the root folder of the PostgreSQL installation. Open `pg-env.bat` and verify the path settings. Save the file if you make changes.

3. Start the server. Navigate to the `bin` folder of the install directory and type: `pg-ctrl start` Enter the password you created for the superuser.

4. Open another terminal window and navigate to the `BW_HOME\samples\binding\rest\BookStore\scripts` folder. Open `readme.txt`. For Unix systems, use the first command in the readme to start the script from the `psql` window. On a Windows machine, copy the second command to start the script from the command line.

5. Navigate to the PostgreSQL `bin` folder and paste the command line into the terminal window. Modify the command as needed. For Windows, use forward slashes in the command. Run the command to create the database, the database tables, and populates the database.

6. Open the PostgreSQL pgAdmin UI utility to see the database and tables.

Creating a New Process

These steps show how to create a new process.

Procedure

1. Open TIBCO Business Studio™.

2. Open the Design perspective by clicking the icon in the upper right.

3. Click the File Explorer tab. If the tab is not visible, click Window > Show View > Other > FileSystem > File Explorer and click OK.

4. Click File > Switch Workspace and select or open a clean new workspace.

5. In the samples directory, select binding > rest > Bookstore and double-click `tibco.bw.sample.binding.rest.BookStore.zip`. This opens the project in the Project Explorer.

6. In the Project Explorer, expand the `tibco.bw.sample.binding.rest.BookStore` project.

7. You can also import the sample using the File > Import > General > Existing Studio Projects into Workspace > Select Archive File > Browse option.

8. The project is displayed in the Project Explorer panel on the left.

   - `tibco.bw.sample.binding.rest.BookStore` Processes
     - `tibco.bw.sample.binding.rest.bookstore`
     - `tibco.bw.sample.binding.rest.bookstore.db`

9. Expand the folders in the project to see all the project processes and resources. Refer to the Application Development guide for information about the folder structure.
10. Expand **Processes** and then expand `tibco.bw.sample.binding.rest.bookstore.db`. You will see **BooksDB.bwp**.

   *bwp is a TIBCO ActiveMatrix BusinessWorks™ process.*

11. Double-click **BooksDB.bwp**.

    The TIBCO ActiveMatrix BusinessWorks™ process comprises:

    - Green chevron on the left indicates the service details.
    - `addBooks`, `getAllBooks`, and so on indicate the operations implemented by this process.
    - Each operation is implemented separately.
12. Double-click an operation to display the process for example, **BooksPersist > addBooks**.
   a) In the addBooks operation, you can see a JDBC activity.
   b) The activity is repeated using a ForEach group.
   c) addBooksOut represents the **Response** to the web service request.

13. To add a new process package named **tibco.bw.sample.rest**, right-click on **Processes** in the Project Explorer and select **New > BusinessWorks Package**.

14. In the BusinessWorks Package screen, specify **tibco.bw.sample.rest** in the **Name** field.
15. Click **Finish** and verify that the new package `tibco.bw.sample.rest` has been added in the Project Explorer.

![View of the Project Explorer](image)

### Building a REST Service

This section details how to build a REST service.

#### Procedure

1. To define a REST Resource named **MyBooks**, select `tibco.bw.sample.rest` > **New** > **BusinessWorks REST Resource**.

![REST Service Wizard](image)

The REST Service Wizard window opens.

2. Specify the following values in the REST Service Wizard window.
   a) **Resource Name**: MyBooks
   b) **Summary**: Summary about the new REST service. (default)
   c) **Resource Service Path**: Auto-filled
   d) **Resource Definition**: Select Browse > Schemas > `Books.xsd` > **Books** in the Select Schema Element Declaration window.
   e) **Operations**: Select POST and GET check boxes.
   f) **Implementation Data**: Accept the default value of **Structured**.

3. Click **Finish**.

This creates a new process **MyBooks.bwp** process is opened in the **Process Editor**.
4. Open the `tibco.bw.sample.binding.rest.bookstore.db` process in the Project Explorer and select the `BooksDB.bwp` process. Drag it to the Process Editor and drop it on the implemented POST operation.

A menu is displayed with two options: Create Invoke Activity and Create Reference and Wire Process.

5. Select Create References and Wire Process.

The references are added to the process. The purple chevron indicates the service and its operations that can be referenced by the process.

6. To update the POST process to invoke the appropriate external service operation:
   a) Click the `addBooks` operation.
b) Select and drag the operation to the left of the **postOut** activity and drop it. An Invoke process activity is created.

7. Click the newly added activity. Select the icon and connect **addBooks** to **postOut**.

8. Click the **getAllBooks** operation and select, drag, and drop the operation to the left of the **getOut** activity in the OUT process.
9. Connect **getAllBooks** to **getOut**.

10. Save your changes.
11. Click the **addBooks** activity and select **Properties > Input**.
12. Expand the data tree in the **Data Source** pane to locate the Book element.
13. Drag the Book element from the left to the Book* element on the right.

14. In the pop-up window, select **Make a Copy of each Book** and click **Finish**.
   
   The **Input** tab will look like this:

   ![Input tab image](image)

15. Save your changes.

16. Click the **postOut** activity and open the **Properties > Input** tab. Expand the **post** activity and drag the Book* element from left to right.

17. In the pop-up window, select the **For each** option and click **Next**. Click **Finish** on the **Auto-Map** window. The **Properties > Input** tab will look similar to this:

   ![Input tab image](image)

18. Click **getAllBooks** and select **Properties > Input**.

19. In the **XPath Expression** pane, add a dummy value to the input element, such as, "Get All Books". The input must be in quotes.

   ![XPath Expression pane](image)

20. Click the **getOut** activity in the **Process Editor**, and select the **Properties > Input** tab. Expand the **getAllBooks** activity and choose Book* to map the Book* element from left to right. In the pop-up window, choose **Make a Copy of each Book** and click **Finish**. The tab will look similar to this:
Result

Your project is complete without any errors.

Testing the REST Service

You can now test the REST service using the built-in tester and the Swagger UI.

Procedure

1. In the Project Explorer, expand the `tibco.bw.sample.binding.rest.BookStore.application` process and expand the Package Unit > Properties folder.

2. In the Properties window, open the `tibco.bw.sample.binding.rest.BookStore` process and set the default ApplicationProfile to match the operating system you are running on. The bracketed profile in the column head is the one that is selected:

   ![Properties Window](image)

   - `HOST`: localhost
   - `DBPassword`: ******
   - `DBPORT`: 8123
   - `DBURL`: jdbc:postgresql://localhost:503
   - `DBNAME`: postgres
   - `DBdriver`: org.postgresql.Driver

3. Provide valid values for the application properties including a valid user name, password, and database URL to connect to your PostgreSQL database if different from the default setting.

4. Verify your JDBC connection.
   a) Expand the Resources folder in the Project Explorer for the `tibco.bw.sample.binding.rest.BookStore` process.
   b) Double-click `JDBCConnectionResource.jdbsResource`.
   c) In the JDBC Driver section of the window, click Test Connection to verify the connection. If you change the JDBC driver folder from the default, click Click Here to Set Preferences and set the JDBC driver folder to the folder where you downloaded PostgreSQL JDBC Driver.
5. Click File > Save.

6. In the Project Explorer, expand the Processes directory if it is not expanded and double-click MyBooks.bwp.

7. Click Run > Debug Configurations.


9. Click the Applications tab and then click the Deselect All button if you have multiple applications. Select the check box next to tibco.bw.sample.binding.rest.BookStore.application.

10. Click Debug. This runs the sample in Debug mode.

   The Console view is opened and shows engine messages similar to: Started BW Application [tibco.bw.sample.binding.rest.BookStore.application:1.0].

11. In the Console view, press Enter to display the prompt: <>@BWEclipseAppNode>

   Enter the OSGi command lrestdoc. This lists the Swagger UI URL as the discovery URL:

12. Launch the Google Chrome browser and open http://localhost:7777/tibco.bw.sample.binding.rest.BookStore.application. Click Books or Events to see the operations. Click MyBooks to see the REST service operations you just added. See the section called Testing the POST and GET Operations for information.
13. Expand the Books and Events headers, and test out the operations as listed below.

**Result**

Click **Books** or **Events** in the Swagger UI to view the following operations for Books and Events:

**Books**

- Post books
- GET books
- GET book by ISBN
- PUT book by ISBN
- DELETE book by ISBN

**Events**

- POST Events
- GET Events
- GET Event by EventID
- PUT Event by EventID
- DELETE Event by EventID

**GET books** returns an output similar to the following:

```json
{
  "Book": [
    {
      "isbn": "0061122416",
      "name": "The Alchemist",
      "description": "Every few decades a book is published that changes the lives of its readers forever. The Alchemist is such a book",
      "authorName": "Paul Coelho",
      "releaseDate": "2006-04-25",
      "vintage": true,
      "signed": true,
      "price": 11.9
    },
    {
      "isbn": "0071450149",
      "name": "The Power to Predict",
      "description": "How Real Time Businesses Anticipate Customer Needs, Create Opportunities, and Beat the Competition",
      "authorName": "Vivek Ranadive",
      "releaseDate": "2006-01-26",
      "vintage": false,
      "signed": true,
      "price": 15.999
    }
  ]
}
```

**GET books** by ISBN returns an output similar to the following for ISBN 0061122416:

```json
{
  "isbn": "0061122416",
  "name": "The Alchemist",
  "description": "Every few decades a book is published that changes the lives of its readers forever. The Alchemist is such a book",
  "authorName": "Paul Coelho",
  "releaseDate": "2006-04-25",
  "vintage": true,
  "signed": true,
  "price": 11.9
}
```
The books.log file is generated with the following information:

```json
POST Books-------->{"Book":[{"isbn":"1451648537","name":"Steve Jobs","description":"Biography of Apple Co-Founder Steve Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-24","vintage":false,"signed":false,"price":21},
{"isbn":"0385537859","name":"Inferno","description":"Robert Langdon returns in Dan Brown's latest fast paced action thriller","authorName":"Dan Brown","releaseDate":"2013-05-14","vintage":false,"signed":true,"price":14.09},
{"isbn":"0399103421","name":"The Godfather","description":"The Godfather is an epic story of a New York's top mafia family, loyalty, and how men of honor live in their own world, and die by their own laws.","authorName":"Mario Puzo","releaseDate":"1969-03-10","vintage":true,"signed":true,"price":50}]
}*************************************************************/

```json
GET Books--------->{"Book":[{"isbn":"1451648537","name":"Steve Jobs","description":"Biography of Apple Co-Founder Steve Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-24+05:30","vintage":false,"signed":false,"price":21},
{"isbn":"0385537859","name":"Inferno","description":"Robert Langdon returns in Dan Brown's latest fast paced action thriller","authorName":"Dan Brown","releaseDate":"2013-05-14+05:30","vintage":false,"signed":true,"price":14.09},
{"isbn":"0399103421","name":"The Godfather","description":"The Godfather is an epic story of a New York's top mafia family, loyalty, and how men of honor live in their own world, and die by their own laws.","authorName":"Mario Puzo","releaseDate":"1969-03-10+05:30","vintage":true,"signed":true,"price":50}]
}*************************************************************/

```json
GET Book By ISBN------->{"isbn":"1451648537","name":"Steve Jobs","description":"Biography of Apple Co-Founder Steve Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-24+05:30","vintage":false,"signed":false,"price":21}*************************************************************/

```json

```json
GET Events By ISBN---->{}*************************************************************/

Testing the POST and GET Operations

An available RESTful service displays the GET operation in the Swagger UI. The POST operation is tested using the JSON service. It is important to test these operations by doing some simple tasks. This section explains how to test the POST and GET operations you just added.

Procedure

1. Click **MyBooks**. It expands and displays the POST and GET operations.

   ![swagger](http://machinename-i4409.7777/tibco.bw.sample.binding.rest.Ba api_key Explore)

2. Click the **POST** icon to display its details.

3. Provide values to the Books parameter. You can use the JSON payload in `BW_HOME\samples \binding\rest\BookStore\samplejson` folder.

4. Click the **Try it out!** button.

5. Now click the **GET** icon to display its details.

6. Click the **Try it out!** button.
The response displays a list of books returned by the REST service from the database.

```
{
    "book": [
    {
        "isbn": "835553709",
        "name": "Inferno",
        "description": "Robert Langdon returns in Dan Brown's latest fast paced action thriller",
        "authorName": "Dan Brown",
        "releaseDate": "2013-09-14T01:38",
        "vintage": false,
        "signed": true,
        "price": 14.99
    },
    {
        "isbn": "8399185421",
        "name": "The Godfather",
        "description": "The Godfather is an epic story of a New York's top mafia family, loyalty, and how men of honor live in their world",
        "authorName": "Mario Puzo",
        "releaseDate": "1993-09-10T01:38",
        "vintage": true
    }
    ]
}
```

7. After you have finished, go back to TIBCO Business Studio™ and click the Terminate icon in the Console view to stop the process.

**Troubleshooting**

Your may encounter some errors while executing or running the process. The following are some of the possible errors you may encounter and their resolutions.

<table>
<thead>
<tr>
<th>Error Encountered</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unable to insert rows into the database using the <code>dbsetup.sql</code> script in the scripts folder.</td>
<td>Use the sample JSON payload from the sample json folder to post the data.</td>
</tr>
<tr>
<td>The REST Swagger UI page is not visible.</td>
<td>Verify that the application has started and that you are accessing the correct URL. Use the <code>lrestdoc</code> command in the Console view to get the Swagger UI URL.</td>
</tr>
<tr>
<td>Problem markers are visible in the project.</td>
<td>Clean the project by invoking Project &gt; Clean or by switching to a clean new workspace.</td>
</tr>
<tr>
<td>Getting the File was not found exception.</td>
<td>Ensure that the <code>books.json</code> and <code>book_put.json</code> files are present at the location described in the Input_File and Input_File_1 module properties.</td>
</tr>
<tr>
<td>The PostgreSQL server does not start.</td>
<td>Make sure you are not running as Administrator.</td>
</tr>
<tr>
<td>The database and database tables are not created.</td>
<td>Open the <code>readme.txt</code> file for the sample, located in the <code>BW_HOME\samples\binding\rest\BookStore\scripts</code> folder. Run the <code>dbsetup.sql</code> script from a command line, not the <code>psql</code> window.</td>
</tr>
</tbody>
</table>
Error Encountered  Resolution

Getting an unregistered user error message while running the process.  Select all the check boxes in the Role Privileges tab in the pgAdmin UI and run the process again. See the image below.

![Image of pgAdmin UI with Role Privileges tab open]
REST Reference

A REST reference is a client process that is used to invoke an external REST service. The service must be running and accessible from the system where the reference resides at the time of its invocation.

REST Reference Binding

REST Binding provides external connectivity for REST over HTTP. You can specify custom HTTP headers and parameters using REST binding. It supports GET, PUT, POST, and DELETE HTTP methods. It also supports JSON, XML, and plain text message types.

Binding

This section has the following fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation Resource or Resource Service Path</td>
<td>The Swagger specification in the project which defines the REST endpoint. This field is visible only when the reference is defined by a Swagger specification, otherwise this field is replaced by the Resource Service Path which is the path to the resource.</td>
</tr>
<tr>
<td>Resource Service Path</td>
<td>Resource path of the REST Resource to invoke</td>
</tr>
<tr>
<td>HTTP Client</td>
<td>The name of the HTTP Client</td>
</tr>
<tr>
<td>Request Client Format</td>
<td>The type of request message format. The two available reply message format options are: JSON and XML Applies to all operations unless overridden at the operation level.</td>
</tr>
<tr>
<td>Reply Client Format</td>
<td>The type of reply message format. The two available reply message format options are: JSON and XML Applies to all operations unless overridden at the operation level.</td>
</tr>
</tbody>
</table>

Summary tab

This tab shows the following details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>A brief description of the operation</td>
</tr>
<tr>
<td>Nickname</td>
<td>The name of the HTTP method, for example, GET, PUT, POST, and DELETE</td>
</tr>
</tbody>
</table>

Request tab

This section shows the following details.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>Client request format, either JSON or XML</td>
</tr>
<tr>
<td>JSON Definition Style</td>
<td>Specifies whether the request item is a single element or an anonymous array</td>
</tr>
<tr>
<td>Request</td>
<td>Data type of the payload, can be primitive data type or complex data type</td>
</tr>
<tr>
<td>Query and Header Parameters</td>
<td>Click the parameter name to edit it. These are the available options:</td>
</tr>
<tr>
<td></td>
<td>• String</td>
</tr>
<tr>
<td></td>
<td>• Integer</td>
</tr>
<tr>
<td></td>
<td>• Boolean</td>
</tr>
<tr>
<td></td>
<td>• Long</td>
</tr>
<tr>
<td></td>
<td>• Float</td>
</tr>
<tr>
<td></td>
<td>• Double</td>
</tr>
<tr>
<td></td>
<td>• Byte</td>
</tr>
<tr>
<td></td>
<td>• Binary</td>
</tr>
<tr>
<td></td>
<td>• Date</td>
</tr>
<tr>
<td></td>
<td>• Date Time</td>
</tr>
<tr>
<td></td>
<td>• Password</td>
</tr>
<tr>
<td></td>
<td>Repeating and Required values can be toggled.</td>
</tr>
</tbody>
</table>

**Response tab**

This tab shows the following details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use HTTP Headers</td>
<td>Select this checkbox only if you require a service to send back a custom</td>
</tr>
<tr>
<td></td>
<td>header or value or if you need to see the response headers. This checkbox is</td>
</tr>
<tr>
<td></td>
<td>checked by default if custom headers are used.</td>
</tr>
<tr>
<td>Format</td>
<td>Response format requested by the client, can be either JSON or XML or both.</td>
</tr>
<tr>
<td></td>
<td>The service must support the format(s) that the client requests for this</td>
</tr>
<tr>
<td></td>
<td>operation</td>
</tr>
<tr>
<td>JSON Definition Style</td>
<td>Specifies whether the response item is a single element or an anonymous</td>
</tr>
<tr>
<td></td>
<td>array</td>
</tr>
<tr>
<td>Resource Schema</td>
<td>Data type of the response being sent back, can be either primitive or complex</td>
</tr>
<tr>
<td></td>
<td>data type</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| Header Parameters   | Click the parameter name to edit it.  
  These are the available types:  
  - String  
  - Integer  
  - Boolean  
  - Long  
  - Float  
  - Double  
  - Byte  
  - Binary  
  - Date  
  - Date Time  
  - Password  
  Repeating and Required values can be toggled. |

**Response Status tab**

This tab shows the following details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>The response code returned</td>
</tr>
</tbody>
</table>
| Type        | These are the available options:  
  - String  
  - Integer  
  - Boolean  
  - XSD element: Select this option to either select the XSD schema element available under the Schemas folder of your project or create a new XML schema resource. |
| Reason Phrase | A brief description of what the code indicates |

**Policies**

To associate a new or existing policy with the REST Binding, click Policies node under the Rest Reference tree, then click the Add Policy button to create a new policy. To edit an existing policy details, click the policy name and edit any of the following fields.
### Creating a REST Reference

The REST reference is used to consume a REST service. The REST client has a reference which it uses to invoke a REST service. You can have multiple REST references to one REST service. References that are created using a Swagger file, cannot be modified in Studio. You can only view the details of such a reference. References that are created using the REST Reference Wizard can be modified in Studio since such references are not associated with a Swagger file which they have to adhere to.

You can create a REST reference in one of the following ways:

- Using a Swagger file
- Using the REST reference wizard
- Using the Debug engine in your Studio

### Using Swagger to Create a REST Reference

If you have the Swagger file, you can import it into TIBCO Business Studio and create a REST reference from it. At the time of the reference creation the service created using that Swagger file may or may not be running or even exist. But when the reference actually invokes the service, at the time of the invocation of the service by the reference, the service must be running and accessible to the reference process. References that are created using an imported Swagger file, cannot be modified in Studio. You can only view the details of such a reference.

### Prerequisites

The Swagger file must be imported into the Service Descriptors folder of the project. For Swagger files on your local machine you can import them by pointing the File Explorer to the location of the Swagger file, then dragging and dropping the file from the File Explorer into the Service Descriptors folder of your project. topic for details.

To create a REST reference from a Swagger file:

1. Expand the .json file in the Service Descriptors folder in Project Explorer.
2. Drag and drop an endpoint in the API from the Service Descriptors folder to the right boundary of the Process Editor.
Using the Wizard to Create a REST Reference

TIBCO Business Studio allows you to invoke a REST service without a Swagger file. As long as you have the URL for the endpoint in a running service, you can use the REST Reference Wizard in TIBCO Business Studio to create a REST reference to invoke the service. A reference created using a wizard can be modified since it does not have a Swagger contract to adhere to. The wizard allows you to choose which operation(s) to implement in the reference as well as set up the data elements used for the request and the response of each operation. You can add additional operations to the reference after the reference has been created too.

Prerequisites

- You must have the endpoint URL
- You must know which operations are supported by the service

To create a REST reference, do the following.

Obtain the URL for the service endpoint

Typically, the URL for an endpoint can be obtained from the API web page or some other documentation where the service is described. This documentation should list the details of the service such as operations that the service supports, the parameters used and the data definitions for the requests and responses.

Verify the service is running

You can verify that the service is running by invoking it using a testing tool such as Postman or SoapUI.

Create a new application and open the REST Reference Wizard

1. Create a new application if you do not already have one. Refer to the section "Creating an Application" in the Application Development Guide.
2. Click the Create REST Reference icon to open the REST Reference wizard.
3. Enter the URL for the service in the Service URL text box.
4. Select the operations that you want to implement in your reference by clicking their checkbox(s) and click Next.
5. Configure the request and response type for your operations in the Configuring the Operation screen by selecting the request and response type from their respective drop-down menus. Click Finish. The wizard creates a REST reference. You should see it on the extreme right of your Process Editor.
6. Open the Reference properties view by clicking on the reference name.
7. Click the **HTTP Client** link to open the HTTP Client Shared Resource configuration page and verify that the **Default Host** has been set to the host where the service resides.

8. Make sure that the port number is correct.
   
   *Do not use port 7777. It is reserved for internal use by ActiveMatrix BusinessWorks.*

9. You can optionally do the following in the Reference properties view:
   
   - Add parameters for your operations
   - Configure operation details, such as Request and Response for an operation by clicking on the operation in the Operations table, then configuring the Response and/or Request details in their respective tabs. See **Parameter Support** section for more details on how to add parameters.
   - Optionally, you can add an operation using the Create REST Operation wizard. To open this wizard, hover your mouse next to the REST reference name until you see the **Create REST Operation** icon and click the icon to open the wizard:

   ![Create REST Operation Wizard](image)

   On the **Create new REST Operations** page, select the operation you want to create and click **Next**. Configure the Request and Response for the operation if need be and click **Finish**.
You can now invoke any operation by dragging and dropping it in the process editor.

REST Reference Wizard

Rest Reference Wizard is used to create a Rest Reference in TIBCO Business Studio which a REST client can use to invoke a running service.

The Rest Reference Wizard has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service URL</td>
<td>The URL to the REST Service.</td>
</tr>
<tr>
<td>Operations</td>
<td>These are the HTTP methods implemented by the REST Reference process to access the REST service. Currently only POST, GET, PUT and DELETE methods are supported.</td>
</tr>
</tbody>
</table>
To add additional operations to an existing REST reference process, click the **Create Rest Operation** icon. This icon appears when you hover over the reference in the Process Editor as follows:

Using the Debug Engine in ActiveMatrix BusinessWorks

You can create a reference for a service that is running on your local machine.

**Procedure**

1. Run the service using the **Run BusinessWorks Application** from the right-click context menu.
2. Type `lrestdoc` in the Console once the service is running.
3. You can use the **Try it out!** button to verify that the service is running.
4. Copy the Discovery URL and paste it into a browser address bar and press Enter. The Swagger page in the browser will display the path to the Swagger file of the running service.
5. Copy the Swagger file URL and paste it into another browser tab address bar and press Enter. The browser displays the contents of the Swagger file.
Now that you have the Swagger file for the service you can create a reference using it. The reference can reside on a machine other than the machine where the service is running. The Swagger file that you just obtained from the running service will contain all the information such as the hostname or the machine where the service is running, the port number and the basepath for the service. The reference you create will reach out to the service using this information from the Swagger file.

### Creating the Reference

To create a reference using the Swagger code that you retrieved from the service, you must first create a Swagger file in the Service Descriptors folder of your process.

Create a process for the reference if you do not already have one. You will need to create a `.json` file in its Service Descriptors folder.

#### Procedure

1. Right-click the Service Descriptors folder of the process and select New > File.
2. Enter a name for the `.json` file and click Finish.
3. Right-click the newly created `.json` file in the Service Descriptors folder and select Open With > Text Editor.
4. Paste the JSON code that you copied from the Swagger file into the Text Editor and save the project.
5. Expand the `.json` file in the Service Descriptors folder to expose the paths.
6. Drag and drop a path from the Service Descriptors folder to the right boundary of the Process Editor to create a reference.
To make your reference application portable

The reference that you created above can only access a service that is running on a reachable network address. You can modify the reference to use a hostname and port that is configurable even after this application is deployed. Using a Module Property for the hostname and/or port will allow that property value to be provided by an administrator that is running the application. The value may also need to be changed if the service is physically moved to another host or port.

Follow these steps to modify the reference such that it can invoke a service from a location determined even after this application is deployed:

Procedure

1. Click on the reference name in the Process Editor to open its properties.
2. Click the Bindings tab.
3. Click the HTTP Client link in the Bindings tab.
4. In the HTTP Client section, click the Choose the field value type button next to the Default Host field text box and select Module Property.
5. Enter host <IP address of machine on which the service is running> in the Default Host text box and click (button).
   It creates a module property called host and gives it the value of the service machine hostname that you provided.
6. Open the module properties for the reference application by double-clicking on <Application Name> > Package Unit > Properties
7. Expand the Application module tree to see the newly created module property.
8. Click the up arrow button to promote the module property to Application property.
   Now the IP address becomes available to be accessed from outside the firewall.

REST Reference Tutorial

The REST reference tutorial shows you how to create a simple REST Invoke to an existing REST Service defined by a Swagger specification.

You cannot convert REST reference to SOAP or vice versa.

Prerequisites

The REST service which you want to invoke must be accessible from the reference process at the time of its invocation.

Creating a New Application

1. Open TIBCO ActiveMatrix BusinessWorks™.
2. Open the Design perspective by clicking the Design icon in the upper right corner.
4. Enter tibco.bw.sample.binding.rest in the Project Name text box. Do not change the remaining default settings. Click Finish. This will create a new application module with an empty process.
5. Obtain the Swagger file from the Swagger UI of the running service.
6. Copy the content and create a new file and call it Books.json.
Importing the JSON File into your Project

1. In the Project Explorer, expand tibco.bw.sample.binding.rest application module.
2. Right-click Service Descriptors and select Import > Import... > General > File System and click Next.
3. In the File system dialog, click the Browse button and browse to the location of the Books.json file.
4. Select the check box next to Books.json in the left pane and click Finish.

Creating the REST Reference

1. In the Project Explorer, expand the tibco.bw.sample.binding.rest Service Descriptors folder completely.
2. Select the /books under Books.json and drag and drop it to the right side of the process in the Process Editor. The references are added to the process. The purple chevron indicates the service and its operations.
3. In the Process Editor, right-click Add Activity > General Activities > Timer. Optionally, you can configure the Sleep activity with IntervalInMillisecond as 3000 in a similar manner and connect the Timer with Sleep.
4. Drag the get operation under the purple chevron and drop it on the right of Timer activity (or Sleep if configured) and connect the Timer activity with the get activity.
5. Drag the post operation under the purple chevron and drop it on the right of the get activity, connect the get activity with the post activity.
6. Right-click the get activity select Show Properties View.
7. In the Properties view, select the Input tab and click Show Check and Repair icon in the icon bar on the upper right corner of the Properties view.
8. Select the check box under Fix and click OK.
9. Click Show Check and Repair icon again. Select the check box under Fix and click OK.
10. Select the post activity and right click and select Show Properties View. In the Properties View, select the Input tab and select Data Source tab.
11. Expand $get in the Data Source tab completely.
12. In the XPath Expression pane, expand the post-input completely.
13. Drag and drop Book* from the Data Source tab to the Book* under post-input in the XPath Expression pane.
15. Click Show Check and Repair icon in the icon bar on the upper right corner of the Properties view.
16. Select the check box under Fix and click OK.
17. Click Show Check and Repair icon again. Select the check box under Fix and click OK.
18. In the Project Explorer, select Books.json under Service Descriptors of tibco.bw.sample.binding.rest.basic application module, and right click Open With > Text Editor and locate the "host" attribute. Make a note of the host name and port number.
19. Expand the Resources folder under the **tibco.bw.sample.binding.rest.basic** application module completely.


21. In the HTTP Client section, change the Default Host and Default Port to the values in the **Books.json** file and check the **Default Confidentiality** check box.

22. Click **File > Save All**.

**Testing the REST Reference**

You can now test the REST service using the built-in tester and the Swagger UI. To do so follow these steps:

1. Click **Run > Debug Configuration**.
2. In the left pane of the **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
3. Click the **Applications** tab and then click **Deselect All** if you have multiple applications. Select the check boxes next to **tibco.bw.sample.binding.rest.basic.application** and **tibco.bw.sample.binding.restapp**.
4. Click **Debug**. This runs the sample in debug mode. The Console view is opened and shows engine messages similar to: `Started BW Application [ tibco.bw.sample.binding.rest.Basic.application:1.0]`

5. In the **Debug** view, expand **BWApplication [BusinessWorks Application] > <launched> BWEclipseAppNode > tibco.bw.sample.binding.rest.Process** and select **get**.

6. In the **JobData** view, you can see the job data of the **get** activity.
Refactoring a REST Service or Reference

TIBCO Business Studio allows you to change the configuration of your REST service even after it is created. The way you update a service configuration depends on whether the service was created from an imported Swagger file or whether the service was created ground up in Studio.

Updating a service created from an imported Swagger file

You can use a Swagger file to either create a service by implementing the contract in it or you can use it to call a service that implements the Swagger file. If the Swagger file you used to create your service or reference gets updated by its provider, you may need to update your implementation of that file in order to avail the changes.

The icon to the left of the Swagger file in the Project Explorer in the TIBCO Business Studio displays an indication that the file has been modified in its original location and the local copy of the file is not in synchronization with its source. You can compare the differences between the two and update your local copy if need be. To do so, follow these steps:

Procedure

1. Right-click the Swagger file under Service Descriptors in the Project Explorer.
2. Select Remote Interface. Check for Differences checks for differences between the imported copy and its original.
   Compare Differences first checks for differences between the imported copy of the Swagger file and its original. If there is a difference, the file will appear in the Synchronize tab and if you double click it there it displays the two files side by side with the differences highlighted.
   Update Local Copy updates the copy of the file in your workspace to match the source. It also regenerated the schema. It updates it in both the service and reference.

No changes are performed for processes that have already been created.

Updating a service that was created ground up in Studio

As you change your implementation of the service or reference, TIBCO Business Studio automatically generates the Swagger file to match your changes.

Changing the Request and Response Schemas

You can make changes to the schema for only those APIs that were created from ground up by you without using a Swagger file. A Swagger file is like a contract that must not be broken. An API created using a Swagger file must implement the Swagger contract exactly and cannot be modified. Hence, for APIs that were created using a Swagger file, you can only view the schemas, but cannot modify them.

If you make schema changes to the APIs that were created ground up by you in TIBCO Business Studio, the Swagger for such APIs automatically gets updated by the Studio.

Updating Configuration

You can edit the configuration for a REST service or a reference that was created from scratch without a Swagger file. You cannot edit the configuration for a service or reference that was created using a Swagger file.

View or edit the configuration for a REST service or reference by following these steps:
Procedure

1. Click the service or reference in the Process Editor.
2. In the General tab of the Properties view, click the link for the service or reference binding.
3. In the Components properties, click the Bindings tab.
4. Click an operation under Operations in the RestService or RestReference tree to edit the configuration of the operation.
   The Summary page for that operation opens on the right. To edit the request, response or response status for the operation, click the respective tabs.

Adding an Operation

For REST services and references that were created without a Swagger file, you can edit their configuration to add or delete operations.

To add an operation after a REST service was created, do the following:

Procedure

1. Click the down arrow to the right of the service and select Create REST Operation.

2. In the REST Service Wizard, select a schema for the operation using the Browse button for Resource Definition.
3. Select an operation you want to add by checking its check box.
4. Optionally, click Next to configure the operation. You can configure the operation after you have added it from the Components page. See Updating Configuration for details.
5. Click Finish.

To add an operation after a REST reference was created, do the following:

Procedure

1. Hover your mouse next to the reference name.
2. Click the Create REST Operation icon.
3. In the REST Reference Wizard, select an operation you want to add by checking its check box.

4. Optionally, click Next to configure the operation. You can configure the operation after you have added it from the Components page. See Updating Configuration for details.

5. Click Finish.

Adding or Deleting Parameters

You can add or edit a parameter for an operation in a REST service or a reference that was created from scratch without a Swagger file. You cannot edit the configuration for a service or reference that was created using a Swagger file.

To add or delete parameters for an operation, do the following:

Procedure

1. Click the service or reference in the Process Editor.
2. In the General tab of the Properties view, click the link for the service or reference binding.
3. In the Components properties, click the Bindings tab.
4. Click an operation under Operations in the RestService or RestReference tree to edit the configuration of the operation.

The Summary page for that operation opens on the right. To edit a parameter in the request or response for the operation, click the respective tabs.
## OSGi Commands to List REST URLs

Use the OSGi command, `lrestdoc`, to list REST and Swagger URLs.

The `lrestdoc` command lists the following discovery URL:

```
<@BWEclipseAppNode> lrestdoc
[Application Name]: tibco.bw.sample.binding.rest.BookStore.application
[Discovery Url]
```

The following are the commands to list endpoints.

```
<@BWEclipseAppNode> lendpoints

[Application Name]: tibco.bw.sample.binding.rest.BookStore.application

<table>
<thead>
<tr>
<th>Endpoint Type</th>
<th>Endpoint URL</th>
<th>CLIENT FORMAT</th>
<th>RESOURCE PATH</th>
<th>HTTP METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST</td>
<td><a href="http://localhost:8123">http://localhost:8123</a></td>
<td>JSON</td>
<td>/books</td>
<td>POST, GET</td>
</tr>
<tr>
<td>REST</td>
<td><a href="http://localhost:8123">http://localhost:8123</a></td>
<td>JSON</td>
<td>/event/{EventID}</td>
<td>GET, PUT, DELETE</td>
</tr>
<tr>
<td>REST</td>
<td><a href="http://localhost:8123">http://localhost:8123</a></td>
<td>JSON</td>
<td>/events</td>
<td>POST, GET</td>
</tr>
</tbody>
</table>

TIBCO ActiveMatrix BusinessWorks™ REST Reference
Exception Handling

Errors (or faults) can occur when executing a process. Fault handlers allow you to catch faults or exceptions and create fault-handling procedures to deal with potential runtime errors in your process definitions.

Fault handlers are the recommended way to catch faults or exceptions in a process. Two types of fault handlers are available: Catch Specific Fault and Catch All Faults.

Fault handlers are defined at the scope level, allowing you to catch faults or exceptions thrown by activities within a scope. To catch faults or exceptions specific to an individual activity, you need to define a new scope for that individual activity and attach a fault handler to the new scope.

At runtime, once a fault handler is executed, the associated scope will not complete due to the error thrown. If a fault is not thrown in the fault handler, the process execution continues with the first activity that follows the scope. If a fault is thrown in the fault handler, then the engine looks for an enclosing scope that is designed to handle the fault. If one is found, the engine executes it. Once the enclosing fault handler finishes its execution, the engine executes the next activity following the scope. If no fault handlers are found in the enclosing scopes, then the job terminates with a fault.

Creating Faults

Fault handlers are used to catch faults or exceptions and create fault-handling procedures to deal with potential errors.

Fault handlers are defined at the scope level allowing you to catch faults or exceptions thrown by activities within a scope. There are two types of fault handlers: Catch Specific Fault and Catch All Faults.

Fault handlers can be defined at the process level, or at a scope level within a process. The diagram below shows two fault handlers - one defined at the process level and the other defined at an inner scope level.

**Fault Handler Attached to an Inner Scope**

![Diagram of fault handler attached to an inner scope]

**Procedure**

1. Select the activities inside the process where the exception is expected to occur and select Create Scope > Scope from the right-click menu.
2. Move the cursor right underneath the scope's lower border to view the icons to create fault handlers.

3. Click on one the following:

- Create Catch to create a fault handler for a specific exception.
- Create Catch All to create a fault handler to catch all exceptions.

A new fault handler is added under the scope.

4. Add activities and configure the fault handling procedure inside the fault handler area. For example, add a Log activity inside the fault handler area to record messages from the exception.

Using the Catch and Rethrow Activities

You can place a Catch block in your process to deal with unhandled exceptions. The Catch block allows you to create a track that handles the exception and proceeds to the end of the current scope; either the end of the process or the end of a group.

You can use the Catch block as an alternative to individually handling exceptions for each activity, or you can use error transitions to handle some exceptions and the Catch block to handle others.

The following figure illustrates the Catch block. The process waits for incoming orders sent by way of HTTP requests. When an order arrives, each line item is checked for availability in the ForEveryLineItem group. If an error occurs while checking the inventory, execution transfers to the CheckInventory activity. A log file entry is written, and then the transition is taken to the end of the group. If the inventory is available, the order is processed, a confirmation email is sent, and the response is sent back to the HTTP client. If the inventory is not available, a response is sent back to the HTTP client stating that one or more items are not available. If an error occurs outside of the ForEveryLineItem group, execution transfers to the CatchAllOthers activity.
Example of using the Catch block

The Catch block can specify the type of exception that should be caught. A list of exceptions that can be raised in the current scope are available on the General tab of the Catch block. Rethrow activity rethrows an exception but to catch it your second catch block, Rethrow activity should be in a group. Any exceptions that are not already handled by an error transition or Catch block can be handled by a Catch block that specifies the Catch All option on the General tab.

The Rethrow activity allows you to throw the exception currently being handled by a Catch path. This is useful if you wish to perform some error processing, but then propagate the error up to the next level.

See TIBCO ActiveMatrix BusinessWorks Bindings and Palettes Reference for more information about the Catch activity and Rethrow activities.

Adding Details to Error Code Descriptions

Error codes are defined in HTTP protocol. Each error code has a certain use that is predefined in the HTTP protocol specification. TIBCO Business Studio allows you to add more details to the description of an error code that might help in debugging the error.

Error codes are defined per operation. So, if the same error code is defined in multiple operations, you can define the error details that are specific to the operation within each operation.

You can edit error codes only in projects that were created ground up in the TIBCO Business Studio. In projects that were created using an imported Swagger file, you can only view the error code details.

To add an error code to an operation or to edit the error code description, follow these steps:

**Procedure**

1. In the Bindings tab of the process, expand the Operations tree in the left pane.
2. Click the operation for which you want to add an error code or modify an existing error code description.
3. Click the Response Status tab in the right pane.
4. Click the **Add Response Status** button (＋) to add an error code. The Reason Phrase that appears by default is the default description for that error code. You can click on it to add more details to the error description.

5. Click the code, type, or reason phrase to edit the field.

**Result**

The newly added error code will be available for you to select.

---

**Using Swagger-Provided Faults**

Fault codes that are defined in a Swagger file that is used in Studio can be viewed from the Response Status tab. View the fault codes used in a REST service or reference follow these steps:
Procedure

1. Click the service or reference in the Process Editor.
2. In the General tab of the Properties view, click the link for the service or reference binding.
3. In the Components properties, click the Bindings tab.
4. Click an operation under Operations in the RestService or RestReference tree to edit the configuration of the operation.
   The Summary page for that operation opens on the right. To view the error codes, click the Response Status for the operation.

Standard HTTP Status Codes

The official registry of HTTP response status codes are maintained by Internet Assigned Numbers Authority (IANA).

You can find a general description of the codes at https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

The following are the most commonly used response codes:

"200", // OK
"201", // Created
"202", // Accepted
"204", // No Content
"301", // Moved Permanently
"303", // See Other
"304", // Not Modified
"307", // Temporary Redirect
"400", // Bad Request
"401", // Unauthorized
"403", // Forbidden
"404", // Not Found
"406", // Not Acceptable
"409", // Conflict
"410", // Gone
"412", // Precondition Failed
"415", // Unsupported Media Type
"500", // Internal Server Error
"503" // Service Unavailable
Security

Currently ActiveMatrix BusinessWorks provides basic authentication and HTTPS support on REST services.

The Basic Authentication policy secures the HTTP layer of REST, SOAP, and pure HTTP services by validating user name and password credentials stored in HTTP headers. User name and password credentials can be authenticated against an XML File Authentication provider.

Overview of Policies

Policies are categorized under the following policy types:

HTTP Security

- Basic Authentication
  The Basic Authentication policy secures the HTTP layer of REST, SOAP, and pure HTTP services by validating user name and password credentials stored in HTTP headers. User name and password credentials can be authenticated against an XML File Authentication provider or an LDAP Authentication provider.

- Basic Credential Mapping
  The Basic Credential Mapping policy enables authentication for specified users by automatically attaching appropriate credentials to request messages before they reach services. You can choose to enforce Fixed or Conditional credential mapping.

SOAP Security

- WSS Provider
  Configure the WSS Provider policy to enforce authentication, confidentiality, integrity, and time stamping of service-side messages.

- WSS Consumer
  Configure the WSS Consumer policy to enforce confidentiality, integrity, time stamping, and credential mapping of response messages.

Managing Policy Resources

Manage policies and policy resources from the ActiveMatrix BusinessWorks Studio.

Creating a Folder for Policies

Policies are always contained in the Policies folder. The folder might not exist in projects you have imported from previous versions of ActiveMatrix BusinessWorks 6.x. If you create a new policy to add to an activity or binding, the Policies folder is automatically created. You can also create a special folder to contain policies.

To create a special folder for policies, follow these steps:

Procedure

1. In the Project Explorer pane, right-click on the application module and select New > Folder to launch the BusinessWorks Application Folder wizard. The Folder wizard opens.
2. Specify the following values in the New Folder window:
• Enter or select the parent folder: Type the name of the parent folder, or select an existing folder to be the parent folder.

• Folder name: Type Policies.

3. Click Finish to create the Policies folder. The new folder displays in the Project Explorer pane.

4. Right-click on the Policies folder, and select Special Folders > Use as Policies Folder.

Result

The folder can now contain policies.

Creating an Authentication Resource

Policies use authentication resources to verify credentials and provide appropriate credentials for users. Follow these steps to create a policy authentication resource.

Procedure

1. In the Project Explorer, right-click on the Resources Folder, and select a new shared resource. For example, select New > XML Authentication.

2. Edit the following fields:

• Resource Folder: Name of the folder where the resource will be located.

• Package: Name of the package in the module where the new resource is added. Accept the default package, or browse to select a different package name.

• Resource Name: Name of the resource. Accept the default name, or type a new name.

3. Click Finish.

Result

The authentication resource displays under the Resources folder in the Project Explorer.

Associating a Policy

Enforce security on your BusinessWorks application, by associating a policy with an existing activity or binding.

Associating a Policy with an Activity

1. In the Process Editor, select the activity to associate the policy with. Activities that support policies display the Policy tab under the Properties tab.

2. From the Properties tab, select the Policy tab.

3. Click the Add Policy to Activity icon.

4. From the Select Policy window, perform one of the following actions:

• Click Create a New Policy to set up a new policy with resources. Policies you can add to the activity are listed under Select the type of policy. For more information about setting up policies and resources from the policy wizard, refer to the appropriate sections under HTTP Security and SOAP Security. Click Finish to create the new policy.
Select an existing policy under Matching Items and click OK.

The policy is associated with the activity.

**Associating a Policy with a Binding**

1. In the Process Editor, select the binding to associate the policy with.
2. From the Properties tab, select the Bindings tab.
3. Click the name of the binding under the Binding section.
4. Click the Bindings tab, and select the Policy field from the tree.
5. Click the Add Policy icon.
6. From the Select Policy window, perform one of the following actions:
   - Click Create a New Policy to set up a new policy with resources. Policies you can add to the activity are listed under Select the type of policy. For more information about setting up policies and resources from the policy wizard, refer to the appropriate sections under HTTP Security and SOAP Security. Click Finish to create the new policy.
   - Select an existing policy under Matching Items and click OK.

The policy is associated with the binding.

**Removing a Policy**

Follow these steps to remove a policy from an activity or a binding.

**Removing a Policy From an Activity**

1. Select the activity associated with the policy.
2. From the Properties tab, select the Policy tab.
3. Select the policy to remove, and click the Delete the selected policy icon.

The policy is no longer associated with the activity.

**Removing a Policy From a Binding**

1. Select the binding associated with the policy.
2. From the Properties tab, select the Binding tab.
3. Under the Policies field, select the policy to remove, and click the Delete the selected policy icon.

The policy is no longer associated with the binding.

**HTTP Security**

Apply security to the HTTP layer of REST, SOAP, and pure HTTP services.
Enforcing Basic Authentication

Implement the Basic Authentication policy to ensure user credentials in request messages are authenticated.

First, set up a new Basic Authentication policy by creating and configuring the policy and its resources. Next, associate the policy with an activity or binding in your application.

Setting Up a Policy with Resources

Follow these steps to set a new Basic Authentication policy with resources:

1. In the Project Explorer, right-click the Policies folder and select New > Policy.

The Policy Wizard opens.
2. Specify the following values in the Create New Policy Window:
   - **Policy Folder**: Name of the folder where policies will be located. Default
   - **Package**: Name of the package in the module where the new policy is added. Accept the default package, or browse to select a different package name.
   - **Policy Name**: Name of the new policy. By default, the policy name is configured to match the security policy you choose. For example, if you select the Basic Authentication policy, the default name of the policy is Basic Authentication.

3. Under **Select the type of Policy**, click **Basic Authentication**.
4. From the **Policy Defaults** drop-down menu, select one of the following options:
   - **Username/Password in workspace XML file**: Select this option to verify user credentials through an XML Authentication resource stored in your workspace. A new Basic
Authentication policy configured for XML authentication and the following resources are produced in your workspace:

- A sample XML File containing user name and password credentials with the default file name XMLUsers.xml
- A new XML Authentication resource with the default file name BasicAuthentication_AuthenticationProvider.authxml

• **Username/Password in filesystem XML file**: Select this option to verify user credentials through an XML Authentication resource stored in your local file system. A new Basic Authentication policy configured for XML authentication is produced in your workspace:

  - A sample filesystem XML File the default file name BasicAuthentication_AuthenticationProvider.authxml

• **Username/Password in LDAP**: Select this option to verify user credentials through an LDAP Authentication resource. A new Basic Authentication policy configured for LDAP authentication and the following resource is produced in your workspace:

  - A new LDAP Authentication resource with the default file name BasicAuthentication_AuthenticationProvider.ldapResource.

• **Empty Policy (No Default)**: Select this option to create a new Basic Authentication policy with no preselected options and no resources.

5. **Optional.** Select **Always create new shared resources** to ensure new resources are generated for the policy and referred to by the policy.

6. **Optional.** Select **Create module properties for common fields** to override default properties in newly created resources with module properties. Resources with module properties for common fields are generated after you select this option.

7. Select **Finish** to create the policy.

**Configuring Resources and the Policy**

For resource configurations, refer to the following topics under the "Shared Resources" topic in the TIBCO ActiveMatrix BusinessWorks Bindings and Palettes Reference guide.

- XML Authentication
- LDAP Authentication

For policy configuration details, refer to the topic "Basic Authentication", under "Policy Resources" in the TIBCO ActiveMatrix BusinessWorks Bindings and Palettes Reference guide.

**Associating the Policy with an Activity or a Binding**

You can associate the Basic Authentication policy with the following activities and bindings:

- HTTP Receiver Activity
- Wait for HTTP Request Activity
  
  Credentials authenticated on this activity are not used for propagation during credential mapping.
- SOAP Service Binding
- REST Service Binding

Refer to **Associating Policies** for instructions on how to enforce a policy on an activity or a binding in your application.
Enforcing Basic Credential Mapping

Map credentials for different types of users by enforcing the Basic Credential Mapping Policy. First, create and configure new policy. Next, associate the policy, with an activity or a binding in your application.

Setting Up a Policy with Resources

Follow these steps to set up a new Basic Credential Mapping policy with resources:

1. In the Project Explorer, right-click the Policies folder and select New > Policy.

![Policy Wizard](image)
2. Specify the following values in the Create New Policy Window:
   - **Policy Folder**: Name of the folder where policies will be located.
   - **Package**: Name of the package in the module where the new policy is added. Accept the default package, or browse to select a different package name.
   - **Policy Name**: Name of the new policy. By default, the policy name is configured to match the security policy you choose.

3. Under **Select the type of Policy**, select **Basic Credential Mapping**.

4. From the **Policy Defaults** drop-down menu, select one of the following options:
   - **Fixed Credentials**: Select this option to ensure a fixed set of credentials are mapped for all users. A new Basic Credential Mapping policy configured for Fixed Basic Credential Mapping and the following resource is produced in your workspace:
     
     ```
     The Policy Defaults menu offers a list of commonly used policy configurations to choose from. After you select a Policy Default, a policy with preconfigured settings and related resources is created. If resources already exist in the module, the newly created policy automatically refers them. However, if no resources exist, new resources with default settings are created and referred to by the policy. Refer to the Default description at the bottom of the Policy Wizard to view policy configurations and new resources that might be created.
     ```
   - **Fixed Credentials**: Select this option to ensure a fixed set of credentials are mapped for all users. A new Basic Credential Mapping policy configured for Fixed Basic Credential Mapping and the following resource is produced in your workspace:
- An Identity Provider resource with the default file name
  BasicCredentialMapping_FixedIdentityProvider.userIdResource

- **Authenticated & Anonymous Users**: Select this option to enforce Basic Credential Mapping for authenticated users and anonymous users. A new Basic Credential Mapping policy configured for conditional basic credential mapping and the following resources are produced in your workspace:
  - An Identity Provider resource for authenticated users with the default file name
    BasicCredentialMapping_AuthIdentityProvider.userIdResource
  - An Identity Provider resource for anonymous users with the default file name
    BasicCredentialMapping_AnonIdentityProvider.userIdResource

- **Role Based Credentials**: Select this option to enforce basic credential mapping for authenticated users with roles. A new Basic Credential Mapping policy configured for conditional basic credential mapping and the following resources are produced in your workspace:
  - An Identity Provider resource for authenticated users with the default file name
    BasicCredentialMapping_AuthIdentityProvider(userIdResource
  - Two separate Identity Provider resources for authenticated users with roles. The default file names of the resources are
    BasicCredentialMapping_RoleIdentityProvider.userIdResource and
    BasicCredentialMapping_RoleIdentityProvider1.userIdResource

- **Empty Policy (No Default)**: Select this option to create a new Basic Authentication policy with no preselected options and no resources.

5. **Optional.** Select *Always create new shared resources* to ensure new resources are generated for the policy and referred to by the policy.

6. **Optional.** Select *Create module properties for common fields* to override default properties in newly created resources with module properties. Resources with module properties for common fields are generated after you select this option.

7. Select **Finish** to create the policy.

**Configuring Resources and the Policy**

For resource configurations, refer to the following topics under the "Shared Resources" topics in the TIBCO ActiveMatrix BusinessWorks Bindings and Palettes guide:

- Identity Provider

Refer to the topic "Basic Credential Mapping", under "Policy Resources" in the TIBCO ActiveMatrix BusinessWorks Bindings and Palettes guide for policy configuration details.

**Associating the Policy with an Activity or a Binding**

You can associate the Basic Credential Mapping policy with the following activities and bindings:

- **SEND HTTP Request Activity**
- **Invoke REST API Activity**

Basic Credential Mapping is not supported on SOAP references. To enforce credential mapping on a SOAP reference, apply the WSS Consumer policy and select either **SAML Token based Credential Mapping** or **Username Token based Credential Mapping**.

Refer to **Associating Policies** for instructions on how to enforce a policy on an activity or a binding in your application.

TIBCO ActiveMatrix BusinessWorks™ REST Reference
The REST and JSON Palette in Studio

The REST and JSON palette in TIBCO Business Studio provides activities that you can use when creating a REST service or reference.

For details on these activities, see the “REST and JSON” palette topics in the Bindings and Palettes Reference section.