TIBCO BusinessWorks™ Container Edition

REST Reference

Version 2.6.3
August 2021

Contents

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 networks allowing applications to share data. REST APIs are used for machine-to-machine communications. REST calls 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, PATCH, 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, PATCH, 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™ for BusinessWorks™ and a JSON object in a Swagger file.

What is an Operation?

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

In TIBCO Business Studio for BusinessWorks, a process implements an operation on a 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 a 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 information about Swagger, see http://swagger.io/.

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

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 a single HTTP Connector to receive responses for all the REST services. From the Swagger UI, select the connector from the options in the drop-down list for **Select HTTP Connector**.

At runtime, the Swagger UI lists 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 are applicable to POST, PUT, and PATCH operations only. They must be defined at the operation level only and cannot be defined at the binding level.

Conversion between Swagger and XML in TIBCO Business Studio for BusinessWorks

When you create a service using a Swagger file, TIBCO Business Studio for BusinessWorks converts the Swagger definitions into XML schema elements. 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 TIBCO Business Studio for BusinessWorks by creating your XML schema using the Schema Editor in TIBCO Business Studio for BusinessWorks.

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 is 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 TIBCO Business Studio for BusinessWorks. If the Swagger file originated in

TIBCO Business Studio for BusinessWorks, then you can modify it in TIBCO Business Studio for BusinessWorks.

TIBCO Business Studio for BusinessWorks maintains a link between the Swagger file and its generated `.xsd` file. Multiple XSD files may be linked to one Swagger file.

Warning: Do not edit a TIBCO Business Studio for BusinessWorks-generated `.xsd` file because its contents are replaced the next time the file is generated.

Not all artifacts in JSON have a direct equivalent in XML. For example, TIBCO Business Studio for BusinessWorks handles Swagger to XML conversion of arrays differently than it handles single elements.

This section explains how TIBCO Business Studio for BusinessWorks 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 for BusinessWorks:

XSD type	Corresponding type in Swagger
long	integer
short, int, integer	integer
double	number
float	number
string	string
base64Binary	string
decimal	number
boolean	boolean
byte	string

XSD type	Corresponding type in Swagger
date	string
dateTime	string
binary	string

Objects

The following table shows how an object in JSON is converted into an XML schema element in TIBCO Business Studio for BusinessWorks. In this example, `Product` is an object that has three attributes called 'product_id', 'description' and 'dispaly_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.

Note: If $ref and type: object are present in the Swagger file, type: object is ignored, and only $ref is considered.

This object in JSON...	...is converted to the following in XSD

```
    "Product": {
                  "type":
"object",

"properties": {

"product_id": {

"type": "string",
                  },

"description": {

"type": "string",
                  },
```
	```
	<xs:element name="Product"
	type="tns:Product"/>
	    <xs:complexType name="Product">
	        <xs:sequence>
	            <xs:element minOccurs="0"
	name="product_id"

	type="xs:string"/>
	            <xs:element minOccurs="0"
	name="description"

	type="xs:string"/>
	            <xs:element minOccurs="0"
	name="display_name"

	type="xs:string"/>
	```

This object in JSON...	...is converted to the following in XSD

`"display_name": {`
 	
`"type": "string",`
`}`
`}`
`}`
	`</xs:sequence>`
	`</xs:complexType>`

Arrays

An array is a collection of identically typed elements. The type can be primitive or complex. For the most part, when TIBCO Business Studio for BusinessWorks 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.

Note: The word "Array" is a key word in TIBCO Business Studio for BusinessWorks. Do not use the "array" suffix in an XSD element name.

Swagger array representation in TIBCO Business Studio for BusinessWorks

When TIBCO Business Studio for BusinessWorks encounters an array in the Swagger file, while generating a schema for it, it 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 TIBCO Business Studio for BusinessWorks-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.

Note: Do not edit a TIBCO Business Studio for BusinessWorks-generated `.xsd` file because its contents are 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.

Array in JSON...	represented in XSD

```
"Products": {
              "type":
"array",
              "items":
{
                "$ref":
"Product"
              }
            }
```
	```
	<xs:element name="**Products**"
	                    type="tns:Products"/>
	    <xs:complexType name="Products">
	        <xs:sequence>
	            <xs:element maxOccurs="unbounded"
	minOccurs="0"
	                        name="Products"
	                        type="tns:Product"/>
	        </xs:sequence>
	    </xs:complexType>
	```

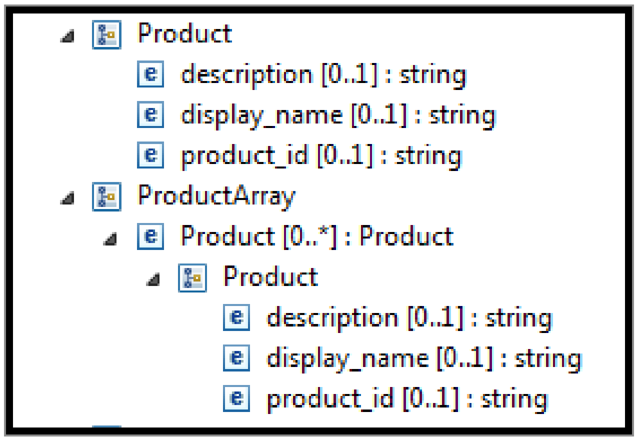
In the example above, `Products` is an array in JSON (denoted by `"type": "array"`) that contains multiple `Product` objects (denoted by `"items":{ "$ref": Product`). The object, `Product`, itself is defined in another location in the Swagger file.

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

The following block in JSON schema...	...gets converted to the following block in XSD

```
"schema": {

"type": "array",

"items": {

"$ref":
"Product"
              }
            }
```
	```
	<xs:element name="ProductArray"
	type="tns:ProductArray"/>
	    <xs:complexType name="ProductArray">
	        <xs:sequence>
	            <xs:element maxOccurs="unbounded"
	minOccurs="0" name="Product" type="tns:Product"/>
	        </xs:sequence>
	    </xs:complexType>
	```

The example above appear as follows in the **Schemas** folder of TIBCO Business Studio for BusinessWorks:


Product
description [0..1] : string
display_name [0..1] : string
product_id [0..1] : string
ProductArray
Product [0..*] : Product
Product
description [0..1] : string
display_name [0..1] : string
product_id [0..1] : string

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 be either structured or anonymous. A structured array is type-based where a type defines the basic construct and 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. JSON uses the concept of anonymous arrays extensively, but the concept 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 example shows JSON payload and its equivalent in XSD in TIBCO Business Studio for BusinessWorks. 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.

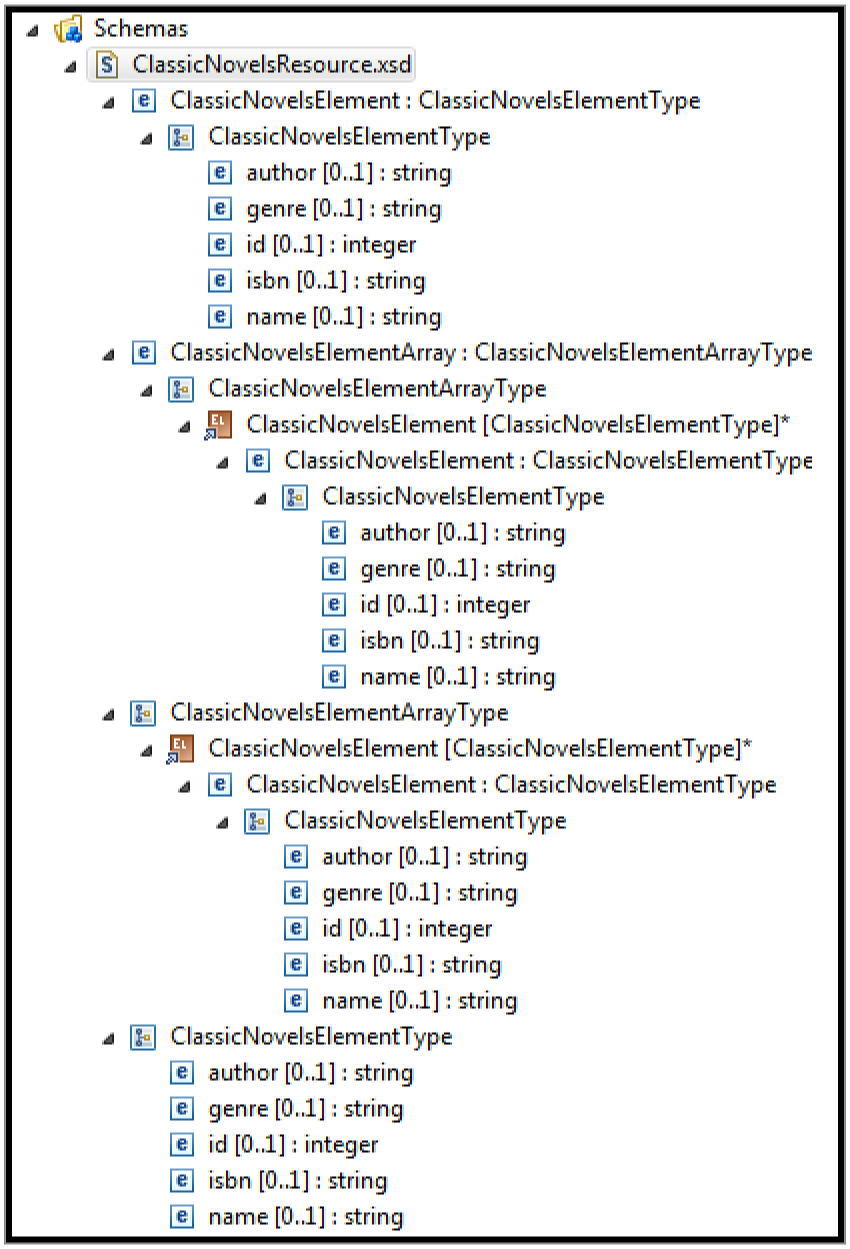
Anonymous array in JSON...	**...gets converted to the following in XSD**

```
[
{
    "id": 1,
    "name": "Great
Expectations",
    "author":
"Charles Dickens",
    "isbn": "13:
```
	```
	    <complexType name="ClassicNovelsElementType">
	      <sequence>
	        <element maxOccurs="1" minOccurs="0"
	name="id" type="integer"/>
	        <element maxOccurs="1" minOccurs="0"
	name="name" type="string"/>
	        <element maxOccurs="1" minOccurs="0"
	name="author"
	               type="string"/>
	```

Anonymous array in JSON...	...gets converted to the following in XSD

```
978-0141439563",
    "genre":
"Classic"
    },

  {
    "id": 2,
    "name": "Jane
Austen",
    "author":
"Emma",
    "isbn": "13:
978-1493663644",
    "genre":
"Romance"
  },
  {
    "id": 3,
    "name": "Jude
the Obscure",
    "author":
"Thomas Hardy",
    "isbn": "13:
978-0140435382",
    "genre":
"Tragedy"
  }
]
```
	```
	      <element maxOccurs="1" minOccurs="0"
	name="isbn" type="string"/>
	      <element maxOccurs="1" minOccurs="0"
	name="genre"
	               type="string"/>
	    </sequence>
	  </complexType>
	  <element name="ClassicNovelsElement"
	           type="tns:ClassicNovelsElementType"/>
	  <complexType
	name="ClassicNovelsElementArrayType">
	    <sequence>
	      <element maxOccurs="unbounded"
	minOccurs="0"
	               ref="tns:ClassicNovelsElement"/>
	    </sequence>
	  </complexType>
	  <element name="ClassicNovelsElementArray"

	type="tns:ClassicNovelsElementArrayType"/>
	```

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



```
Schemas
  ClassicNovelsResource.xsd
    ClassicNovelsElement : ClassicNovelsElementType
      ClassicNovelsElementType
        author [0..1] : string
        genre [0..1] : string
        id [0..1] : integer
        isbn [0..1] : string
        name [0..1] : string
    ClassicNovelsElementArray : ClassicNovelsElementArrayType
      ClassicNovelsElementArrayType
        ClassicNovelsElement [ClassicNovelsElementType]*
          ClassicNovelsElement : ClassicNovelsElementType
            ClassicNovelsElementType
              author [0..1] : string
              genre [0..1] : string
              id [0..1] : integer
              isbn [0..1] : string
              name [0..1] : string
    ClassicNovelsElementArrayType
      ClassicNovelsElement [ClassicNovelsElementType]*
        ClassicNovelsElement : ClassicNovelsElementType
          ClassicNovelsElementType
            author [0..1] : string
            genre [0..1] : string
            id [0..1] : integer
            isbn [0..1] : string
            name [0..1] : string
    ClassicNovelsElementType
      author [0..1] : string
      genre [0..1] : string
      id [0..1] : integer
      isbn [0..1] : string
      name [0..1] : string
```

Forms

TIBCO Business Studio for BusinessWorks supports the use of form parameters as the media type in REST requests for POST, PUT, and PATCH operations. This is the only media type that can be used to transmit binary data and files. Form parameters must be defined at the operation level only and cannot be defined at the binding level.

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 text, two elements get created in TIBCO Business Studio for BusinessWorks:
    - **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:

Data types	JSON	XSD

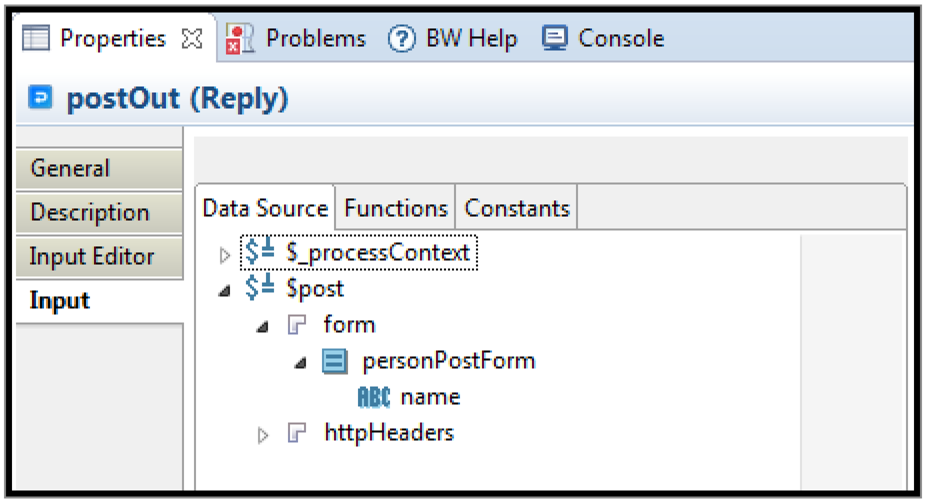
primitive data types	`{`
	`    "/person" : {`
	`      "post" : {`
	`        "description" :`
	`"",`
	`        "operationId" :`
	`"post-person",`
	`        "consumes" : [`
	`"application/x-www-form-`
	`urlencoded" ],`
	`        "produces" : [`
	`"application/json" ],`
	`        "parameters" : [ {`
	`          "name" : "name",`
	`          "in" :`
	`"formData",`
	`          "description" :`
	`"",`
	`          "type" :`
	`"string",`
	`          "required" :`
	`true`
	
	`        } ]`
		`<xs:element`
		`name="personPostForm">`
		`    <xs:complexType>`
		`      <xs:sequence>`
		`                <xs:element`
		`maxOccurs="1" minOccurs="1"`
		
		`name="name" type="xs:string"/>`
		`            </xs:sequence>`
		`    </xs:complexType>`
		`  </xs:element>`

Data types	JSON	XSD

binary or file type	`{`
	`    "/person" : {`
	`      "post" : {`
	`        "description" :`
	`"",`
	`        "operationId" :`
	`"post-person",`
	`        "consumes" : [`
	`"`**`multipart/form-data`**`" ],`
	`        "produces" : [`
	`"application/json" ],`
	`        "parameters" : [ {`
	`          "name" :`
	`"photo",`
	`          "in" :`
	`"formData",`
	`          "description" :`
	`"",`
	`          "type" : "file",`
	`          "format" :`
	`"binary",`
	`          "required" :`
	`false`
	
	`        } ]`
		`<xs:element`
		`name="personPostForm">`
		`    <xs:complexType>`
		`      <xs:sequence>`
		`                <xs:element`
		`maxOccurs="1" minOccurs="0"`
		`name="photo">`
		`          <xs:complexType>`
		`            <xs:sequence>`
		`              <xs:element`
		`maxOccurs="1" minOccurs="0"`
		`name="filename"`
		
		`type="xs:string"/>`
		`              <xs:element`
		`maxOccurs="1" minOccurs="0"`
		`name="content"`
		
		`type="xs:base64Binary"/>`
		`            </xs:sequence>`
		`          </xs:complexType>`
		`        </xs:element>`
		`            </xs:sequence>`
		`    </xs:complexType>`
		`  </xs:element>`

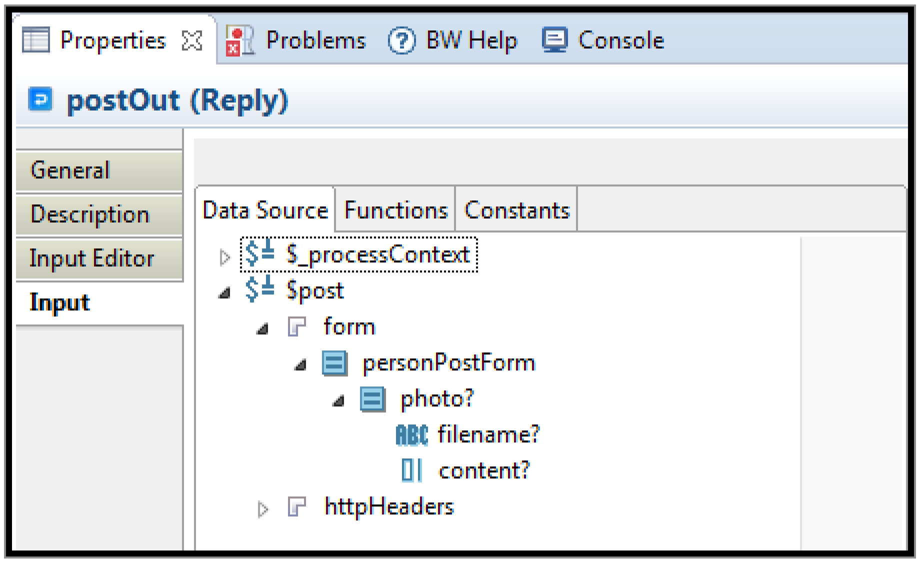
The examples appear as follows in TIBCO Business Studio for BusinessWorks :

Primitive type:



```
▷ $_processContext
◢ $post
    ◢ form
        ◢ personPostForm
            name
    ▷ httpHeaders
```

Binary or File type:



```
▷ $_processContext
◢ $post
    ◢ form
        ◢ personPostForm
            ◢ photo?
                filename?
                content?
    ▷ httpHeaders
```

Working with Path and Query Parameters

TIBCO BusinessWorks™ Container Edition REST APIs support path and query parameters. You can apply path parameters only at the root level and not at individual operation level. You can apply query parameters at root as well as 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>*.

Note: Path parameters that are not immediately enclosed in forward slashes are supported. For example, the parameter `authorName('{isbn}')` in the resource service path `/book/authorName('{isbn}')/` is not directly contained by forward slashes, but still passes successfully.

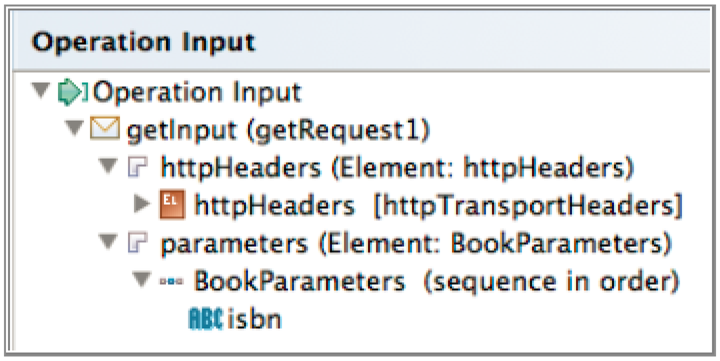
Insert a question mark **(?)** after a parameter to add query parameters to a resource. In the following example, `isbn` is defined as a query parameter, instead of the path parameter, in the resource service path:

```
/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 **Input** for every operation. See the following image for the `/book/{isbn}` example to see the **Input** for the GET operation.



Operation Input

- Operation Input
  - getInput (getRequest1)
    - httpHeaders (Element: httpHeaders)
      - httpHeaders [httpTransportHeaders]
    - parameters (Element: BookParameters)
      - BookParameters (sequence in order)
        - isbn

Important: 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` cannot be used as query or path parameter name.

Important: Do not use the word "fields" as the name of a query parameter. The word "fields" is a reserved keyword for internal use. 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 TIBCO Business Studio for BusinessWorks 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 edits.

   The newly created path parameter appears in the Path Parameters table. You can edit the name of the parameter by clicking on it. You can change its type by clicking it, and then selecting a new type from its drop-down menu.

   Caution: Be aware that these buttons are disabled if you are viewing operations for an API that was created outside TIBCO Business Studio for BusinessWorks and imported into TIBCO Business Studio for BusinessWorks.

Adding or Editing Query Parameters

You can create or modify query parameters for REST services that were created in the TIBCO Business Studio for BusinessWorks 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 () 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 are disabled if you are viewing operations in an API that was created outside TIBCO Business Studio for BusinessWorks and imported into TIBCO Business Studio for BusinessWorks.
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 for BusinessWorks supports sending and receiving arrays in REST requests and responses.

In TIBCO Business Studio for BusinessWorks, you can create a REST API by starting with a Swagger-compliant JSON file or you can create the API from scratch using the wizards in TIBCO Business Studio for BusinessWorks. If you use JSON as your starting point, TIBCO Business Studio for BusinessWorks generates an XSD file when the API gets created. When generating the XSD for an array, TIBCO Business Studio for BusinessWorks creates a wrapper element with an "Array" suffix and another single element containing the definition for the element type contained in the array.

For more information about how TIBCO Business Studio for BusinessWorks handles arrays, see the Arrays section in the topic Conversion Between JSON and XML in TIBCO Business Studio for BusinessWorks .

Note: Do not edit the .xsd file that is generated by TIBCO Business Studio for BusinessWorks.

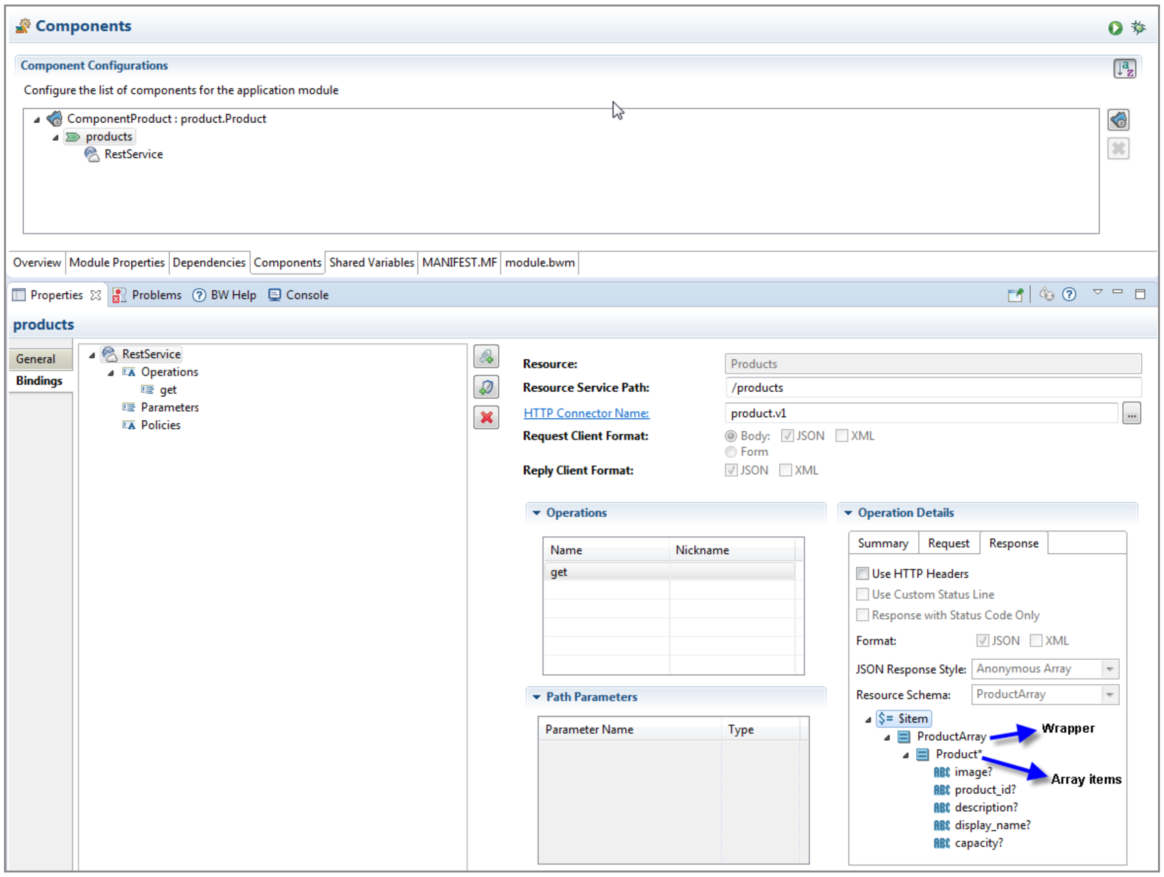
Note: The word "Array" is a key word in TIBCO Business Studio for BusinessWorks. Do not use the "array" suffix in an XSD element name.

Configuring an array in REST binding in TIBCO Business Studio for BusinessWorks

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

Note: 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:


Components

Component Configurations

Configure the list of components for the application module

ComponentProduct : product.Product
products
RestService

Overview | Module Properties | Dependencies | Components | Shared Variables | MANIFEST.MF | module.bwm

Properties | Problems | BW Help | Console

products

General
Bindings

RestService
Operations
get
Parameters
Policies

Resource: Products
Resource Service Path: /products
HTTP Connector Name: product.v1
Request Client Format: Body: ✓ JSON XML; Form
Reply Client Format: ✓ JSON XML

Operations

Name	Nickname
get	

Path Parameters

Parameter Name	Type

Operation Details

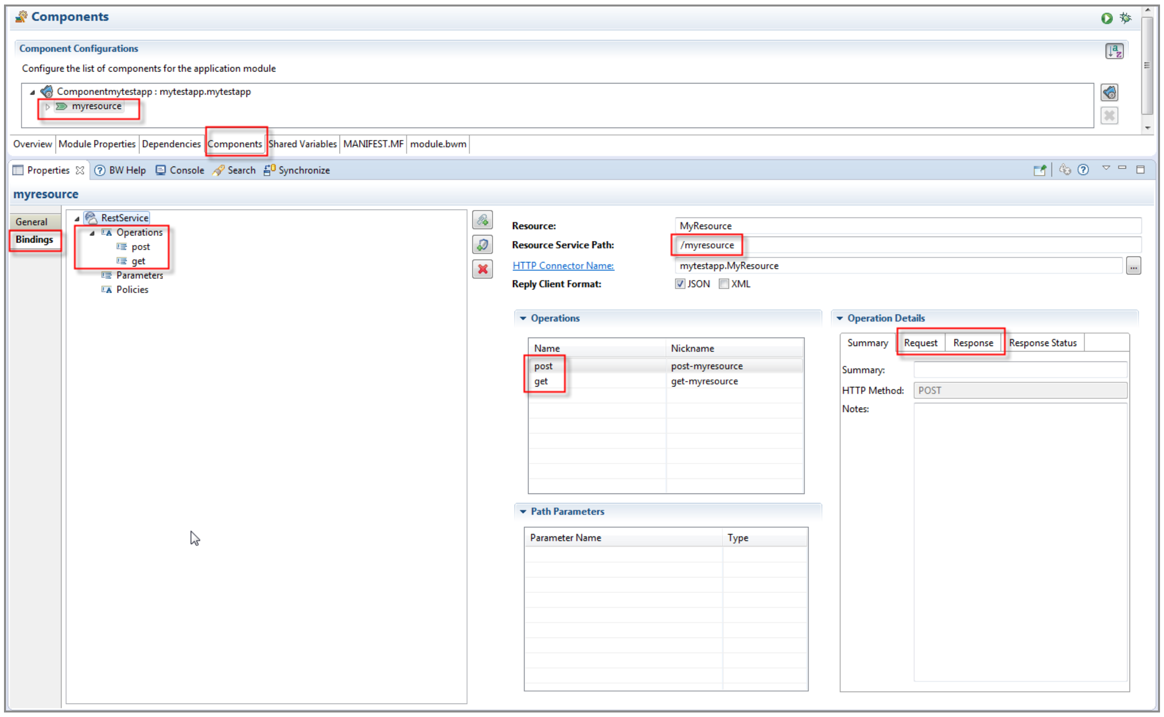
Summary | Request | Response

Use HTTP Headers
Use Custom Status Line
Response with Status Code Only
Format: ✓ JSON XML
JSON Response Style: Anonymous Array
Resource Schema: ProductArray

$item
ProductArray
Product*
image?
product_id?
description?
display_name?
capacity?

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.



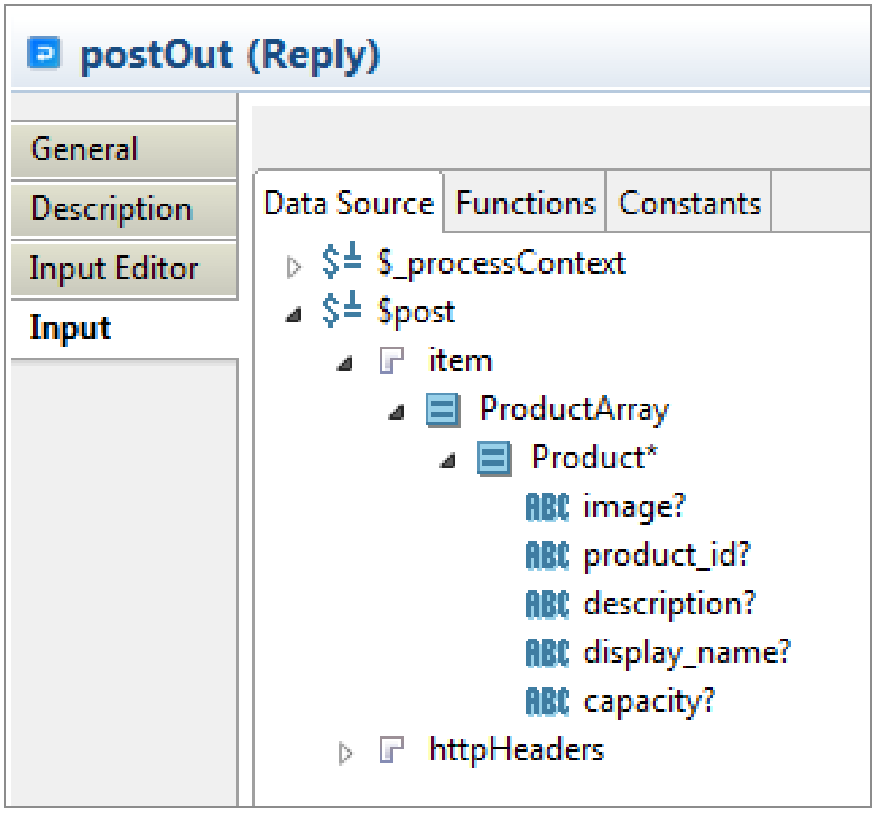
Componentmytestapp : mytestapp.mytestapp
myresource

Resource: MyResource
Resource Service Path: /myresource
HTTP Connector Name: mytestapp.MyResource
Reply Client Format: [✓ JSON] XML

Name	Nickname
post	post-myresource
get	get-myresource

Summary:
HTTP Method: POST
Notes:

4. In the **Operations Details** box, click the **Response** tab.
5. Select the **JSON** check box 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** drop-down 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 for BusinessWorks supports the use of form parameters as the media type in REST requests for POST, PUT, and PATCH operations. This is the only media type that can be used to transmit files.

For more information about how form parameters are represented in JSON and XSD, see the Conversion Between JSON and XML in TIBCO Business Studio for BusinessWorks.

Form parameters are applicable to POST, PUT, and PATCH operations only. You must define them at the operation level only and not at the binding level.

An operation can have one of the following encoding. 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 are created in TIBCO Business Studio for BusinessWorks:

- **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, PUT, and PATCH operations only. You can create or modify parameters for REST services that were created ground up in the TIBCO Business Studio for BusinessWorks without a Swagger file. For REST services that were created using a Swagger file that was imported into TIBCO Business Studio for BusinessWorks 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 () on the right side of the **Form Parameters** table to create a new form parameter.

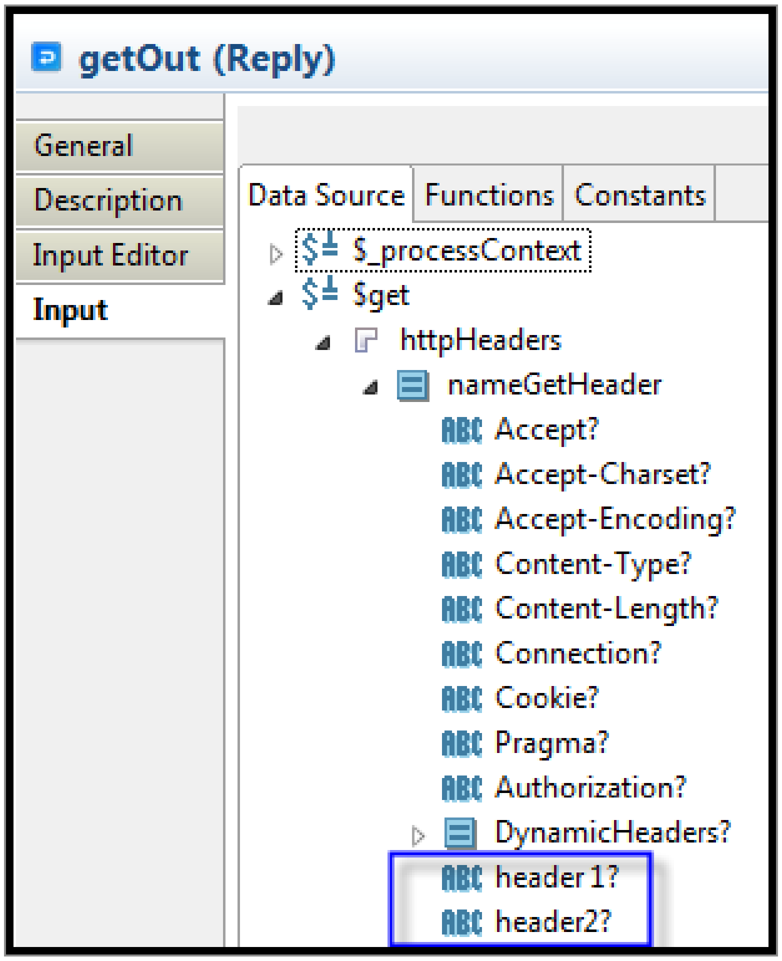
9. Click the name and edit it. To change the type, click the type in the **Type** column and select a type from the drop-down list. Similarly, you can specify if this parameter is required or not by clicking in the **Required** column.

Working with Header Parameters

Header parameters are used for 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 are passed into the request but are not visible to you.

Note: 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 are not editable in TIBCO Business Studio for BusinessWorks. You cannot modify existing parameters or create new ones. You can only view the parameters. Any custom header parameters you create are visible in the **Data Source** tab of TIBCO Business Studio for BusinessWorks as follows:



```
▷ $_processContext
◢ $get
    ◢ httpHeaders
        ◢ nameGetHeader
            Accept?
            Accept-Charset?
            Accept-Encoding?
            Content-Type?
            Content-Length?
            Connection?
            Cookie?
            Pragma?
            Authorization?
          ▷ DynamicHeaders?
            header1?
            header2?
```

You can view or add header parameters in TIBCO Business Studio for BusinessWorks if the Swagger file for your project was auto-generated in TIBCO Business Studio for BusinessWorks by implementing a REST service from XSD.

To add a custom header parameter, follow these steps:

Procedure

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 are disabled if you are viewing operations for an API that was created outside TIBCO Business Studio for BusinessWorks and imported into TIBCO Business Studio for BusinessWorks.

   > **Note:** Do not use a space in the HTTP header name. If you do, the following error is displayed: `Bad message- 400 - Illegal character.`

   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.

7. Use the **Response** tab to add a header parameter to the operation response.

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 for BusinessWorks 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 TIBCO Business Studio for BusinessWorks or imported into TIBCO Business Studio for BusinessWorks 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
- Using the respective wizard in TIBCO Business Studio for BusinessWorks

Supported Message Formats

supports the following message formats: JSON, XML and Text.

Each operation (POST, GET, PUT, PATCH, and DELETE) can have JSON, XML, or Text request or reply settings, independent of the other operations.

If all of the operations have the same setting, then you can change the request or response types at the binding level to change all operations at once. If one or more operations has a different value than the remaining operations, then you cannot make a change at the binding level. You must change the settings for each one independently. This gives you full control over the format of each operation.

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 ([[...], [...]])
- Null type properties throw the XSD schema conversion error in TIBCO Business Studio for BusinessWorks. To create a REST Service or REST Binding edit the input JSON file and change the data type from null to string.
- If a Swagger 3.0 JSON file contains multiple servers URL, then on creating the REST service or a reference from it, the first URL is picked up to configure the shared resource's host and port. You can re-configure the host and port based on the server URL you want to invoke. Change the configurations accordingly in the shared resource.

Not currently supported

- Binary content in JSON as a special case.
- The Swagger 3.0 JSON file containing Open API 3.0 specific features which are not in parity with Swagger 2.0 JSON file.

Restrictions on XML Schema

This topic lists the restrictions on XML Schema.

General Restrictions

- No wildcards or attribute wildcards. For example, `any` element and `any` attribute is not supported.

- Complex types might not contain both an attribute and a child element with the same local name.
- Complex types might not be of the pattern "simple type plus attributes".
- Complex types might 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 might not have `maxOccurs > 1` (same as the restriction on 'all' in the schema specification).
- Substitution groups are not supported.
- Element of simple type with an attribute is not supported.
- The `elementFormDefault` can only be qualified for schemas used by REST binding and JSON activities.
- Schemas should not contain cyclic dependencies within same schema, or on the other schemas.
- Schemas should not have a type that has two child members with the same local name, but different namespaces.
- For float and double values, XML schema always shows exponential values of type `1.0E0`

Using Swagger 1.2 Files

You can use Swagger 1.2 API declaration files.
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 for BusinessWorks provides utilities to convert objects from JSON to XSD elements.

Creating a New XML Schema File

Using TIBCO Business Studio for BusinessWorks you can create a new XML schema file in your project.

Before you begin

The project must exist in the Project Explorer.

Procedure

1. Right-click the **Schemas** folder in your project in the **Project Explorer**.
2. Select **New > XML Schema File**.
3. Enter a file name and click **Finish**.

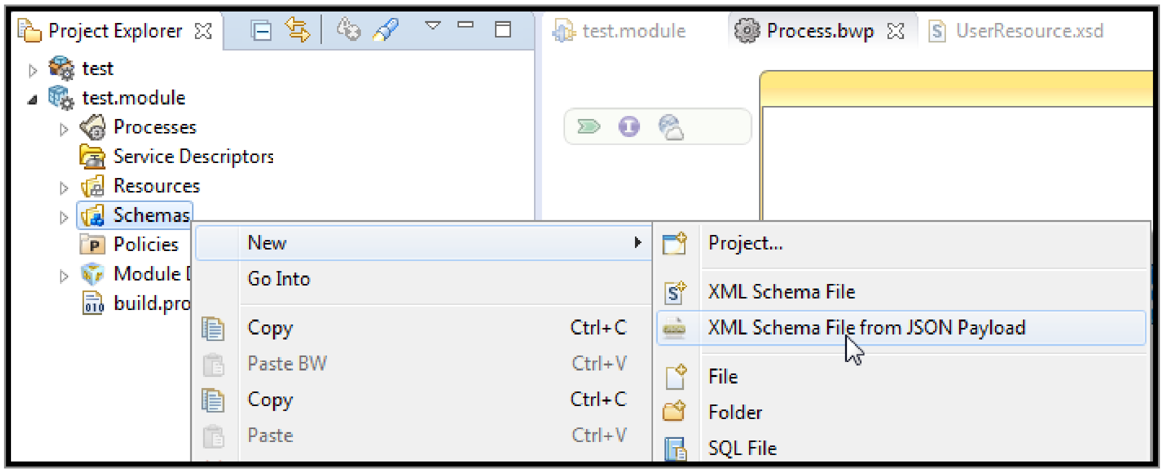
   The file opens in the XML Schema Editor.
4. Right-click anywhere in this editor and click **Add Element** to add a new element.

Creating an XML Schema from a JSON Payload

TIBCO Business Studio for BusinessWorks provides a utility 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 can 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 the schema file and select **Open With > XML Schema Editor**).

4. Save the project.

Generating an XML Schema from a Swagger File

You can generate an XML schema from a Swagger 1.0, 2.0, and 3.0 file using the menu options without creating a service.

Note: This option is enabled only when the JSON file is any of the Swagger 1.0 file, Swagger 2.0 file, Swagger 3.0 file, and has Swagger Definition.

For Swagger files that do not contain Swagger definition, the **Generate XSD Schema** menu option is disabled.

Before you begin

A Swagger 2.0 or a Swagger 3.0 file should exist in the **Service Descriptors** folder of the project. If not, then 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 `.xsd` file gets created in the **Schemas** folder of your project.

2. Save the project.

- To figure 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 contains only those elements that are common to both files. It does 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 POST, GET, PUT, PATCH, and DELETE HTTP methods. It also supports JSON, XML, and plain text message types.

Binding

This section has the following fields.

Field	Description
Resource	The name of the resource.
Resource Service Path	Specify the path to the Service Resource. 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>*. **Note:** Path parameters that are not immediately enclosed in forward slashes are supported. For example, the parameter `authorName('{isbn}')` in the resource service path `/book/authorName('{isbn}')/` is not directly contained by forward slashes, but still passes successfully. If an application contains multiple REST bindings, make sure that the location of the path parameters is unique for each REST binding. An example is that of one REST binding using the `/book/{isbn}` path and

Field	Description
	another REST binding is using the `/book/{authorid}` path. Since `{isbn}` and `{authorid}` are defined at the same location in the URI, one of these services do not function as expected. In addition to path parameters, the path in a REST binding can also contain query parameters. For example, `/resource/path/{pathparam}? query={queryparam}` or `/resource/path/{pathparam}?{ queryparam}`
HTTP Connector Name	The name of the HTTP Connector. **Tip:** To display details about the HTTP Connector resource, click on the **HTTP Connector Name** field. By default, a new **HTTP Connector** shared resource is created when you create a new REST Service binding. Change the field value type to **Module Property** to specify a module property that has been defined as an **HTTP Connector** shared resource.
Response Client Format	The type of response message format. The supported response message formats are: • JSON • XML
Enforce BW Service Response	Select the check box to set the response preference to **BW Service Response**. By default, the check box is not selected, and the response preference is set to the **Accept Header** response. For more information about the REST Service responses based on the Accept Header settings, see the Accept Header Responses topic.
Start Job on Input Exception	Select the check box to start the job when there is a wrong or erroneous input.

Note: Imported projects display the **Authenticate** check box under the Binding section if the check box was selected in a previous version TIBCO BusinessWorks Container Edition. Authentication also remains enabled on the **REST Service Binding** if you do not clear the check box. Clearing the **Authenticate** check box causes a warning message to be displayed 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.

Operations

This section shows the following details.

Field	Description
Name	The name of the HTTP method used, for example, POST, GET, PUT, PATCH, and DELETE.
Nickname	The specified name of the service, for example, getBooks.

Operation Details

This section has the following tabs.

Summary tab

Field	Description
Summary	The summary of the REST resource.
HTTP Method	Displays the HTTP Method specified in the **Operations** section. These are the available HTTP methods: • POST • GET • PUT • DELETE

Field	Description
	• PATCH
Notes	Additional information about REST resource.

Request tab

Field	Description
Use Null for Empty Values	Select the check box to set NULL values instead of empty values in JSON. That is, use NULL values instead of square brackets ([]). By default, the check box is clear.
Ignore Additional JSON Fields	Select this check box to ignore additional fields that are generated due to changes in the external payload when processing the schema. By default, the check box is clear.
Format	Supported formats for REST service request are: • JSON • XML • Text
JSON Definition Style	Select one of the following options: • **Single Element**: Returns an element of corresponding data type or a single schema element when a schema is selected. • **Anonymous Array**: Returns a JSON array without the parent element, where the root element has exactly one child of the type Array. • **JSON with Root**: Includes the root element in the input JSON string.
Request	Data type of the Payload. It can be one of the following: • XSD Element • String

Field	Description
	• Integer • Boolean • Form Data - Tag/Value (application/x-www-form-urlencoded) • Form Data - Multipart (application/form-data)
Query and Header Parameters	You can perform the following operations: • Add Query Parameter • Add Header Parameter • Remove Parameter • Scroll Up • Scroll Down This pane has four columns: • Parameter Name Name of the parameter. Users can edit the parameter name by clicking on the parameter added. • Type Data type of the parameter. It can be: ◦ String ◦ Integer ◦ Long ◦ Float ◦ Double ◦ Boolean ◦ Byte ◦ Binary ◦ Date

Field	Description
	◦ Date Time ◦ Password • Repeating This field can be toggled to Yes and No. • Required This field can be toggled to Yes and No.

Response tab

Field	Description
Use HTTP Headers	Selecting this check box includes the response headers element. Response headers are not commonly used, so select this check box only when you need to include response headers. When you select this check box, you can add custom HTTP fault headers defined in the **Response Status** tab.
Use Custom Status Line	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.
Response with Status Code Only	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.
Use Empty Values for Null	Select the check box to set empty values instead of NULL values in JSON. That is, use square brackets ([]) instead of NULL. By default, the check box is clear.
Format	Supported formats for REST service request are: • JSON • XML

Field	Description
	• Text
JSON Definition Style	Select one of the following options: • **Single Element**: Returns an element of corresponding data type or a single schema element when a schema is selected. • **Anonymous Array**: Returns a JSON array without the parent element, where the root element has exactly one child of the type Array. • **JSON with Root**: Includes the root element in the input JSON string.
Resource Schema	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: • String • Integer • Boolean • XSD element: 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. **Note:** Make sure that the schema resource you select does not contain cyclic dependencies on other schemas , or a type that has two child members with the same local name, but different namespaces.
Header Parameters	This field is enabled only when **Use HTTP Headers** check box is selected. You can perform following operations: • Add Header Parameter • Remove Parameter • Scroll Up • Scroll Down

Field	Description
	This pane has four columns: • Parameter Name Name of the parameter. Users can edit the parameter name by clicking on the parameter added. • Type Data type of the parameter. It can be: ◦ String ◦ Integer ◦ Long ◦ Float ◦ Double ◦ Boolean ◦ Byte ◦ Binary ◦ Date ◦ Date Time ◦ Password • Repeating This field can be toggled to Yes and No. • Required This field can be toggled to Yes and No.

Response Status tab

Column Name	Description
Code	These are unique numbers. Click on the error code to customize it.

Column Name	Description
	Note: Use custom status code 200 only when the response is not defined, that is, when the **Response with status code only** check box is selected in the **Response** tab.
Type	Data type of the error code. Following types are supported: • 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. • String • Integer • Boolean The default type is String.
Reason Phrase	Description of the error code. Click on the value to customize the description.

Path Parameters

This section shows the following details.

Parameter Name	Type
Parameter name of the operation used	The parameter type. It can be any one of the following: • String • Integer • Boolean • Long • Float • Double

Parameter Name	Type
	• Byte • Binary • Date • DateTime • Password

Advanced Configuration

This section has the following field:

Field	Literal Value/ Module Property	Description
Blocking Queue Size	Yes	This field sets the number of threads to be created for a REST service. It gives you control over the number of threads that are created for the REST service. By default, it is set to a large integer value.

Accept Header Responses

REST Service Binding allows users to set preferences for responses. Responses can be set to either **BW Service Response** to honor the TIBCO BusinessWorks™ Container Edition service responses or **Accept Header**, which accepts headers set by the client or browser.

The following table shows the REST service responses based on the **Accept Header** settings.

Accept Header (Client)	BW Service Response	Support till version TIBCO ActiveMatrix BusinessWorks™ 6.3.1	Support from version ActiveMatrix BusinessWorks™6.3.2 to vesion 6.3.5	Support from version ActiveMatrix BusinessWorks 6.4.0 (Default behaviour)	Version ActiveMatrix BusinessWorks 6.4.0 (Enforce BW Service Response)
application or JSON	JSON	JSON	JSON	JSON	JSON
application or XML	JSON	XML	Error: 406	Error: 406	Error: 406
/	JSON	JSON	JSON	JSON	JSON
application or XML;q=0.9	JSON	JSON	Error: 406	Error: 406	Error: 406
application or XML;q=0.9,*/*	JSON	JSON	XML	XML	JSON
application or XML,*/*	JSON	JSON	XML	XML	JSON
application or json,text/plain,*/*	JSON	JSON	NA	text/plain	JSON
application or json,text/plain	JSON	JSON	NA	text/plain	JSON
application/json	XML	text/plain	Error:406	Error:406	Error:406
application/xml	XML	XML	XML	XML	XML
/	XML	XML	XML	text/plain	XML
application/xml;q=0.9	XML	text/plain	XML	XML	XML
application/xml;q=0.9,*/ *	XML	XML	XML	XML	XML

Accept Header (Client)	BW Service Response	Support till version TIBCO ActiveMatrix BusinessWorks™ 6.3.1	Support from version ActiveMatrix BusinessWorks™6.3.2 to vesion 6.3.5	Support from version ActiveMatrix BusinessWorks 6.4.0 (Default behaviour)	Version ActiveMatrix BusinessWorks 6.4.0 (Enforce BW Service Response)
application/xml,*/*	XML	XML	XML	XML	XML
application/json,text/plain,*/*	XML	XML	XML	text/plain	XML
application/json,text/plain	XML	text/plain	Error:406	Error:406	Error:406

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. TIBCO BusinessWorks Container Edition supports the following HTTP methods: GET, PUT, DELETE, PATCH, 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 for BusinessWorks 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 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 the `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.

You can use the `Refactor` option to delete both REST service and reference Bindings WSDL operations using a top down or a bottom up approach.

To create a REST service from a Swagger file:

Procedure

1. Create an empty project. For more information, see "Developing a Basic Process" in *TIBCO BusinessWorks™ Container Edition Concepts*.
2. Import the Swagger JSON file into the **Service Descriptors** folder of your project by dragging 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 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 for BusinessWorks 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, PATCH, DELETE, OPTIONS, HEAD, and custom.

Before you begin

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.

Important: To generate Swagger 3.0 compliant services, in TIBCO Business Studio for BusinessWorks select **Window > Preferences > BusinessWorks > Bindings > REST**. Select the **Swagger Version** as 3.0 from the drop-down list.

By default, the **Swagger Version** is 1.2.

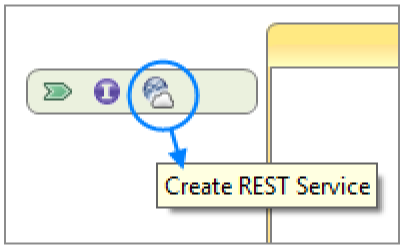
To create a new schema file in TIBCO Business Studio for BusinessWorks:

1. In Project Explorer, right-click the **Schemas** folder.
2. Select **New > XML Schema File**.
3. In the Schema Editor, right-click in the respective box to add a directive, element, type, attribute, or group.

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

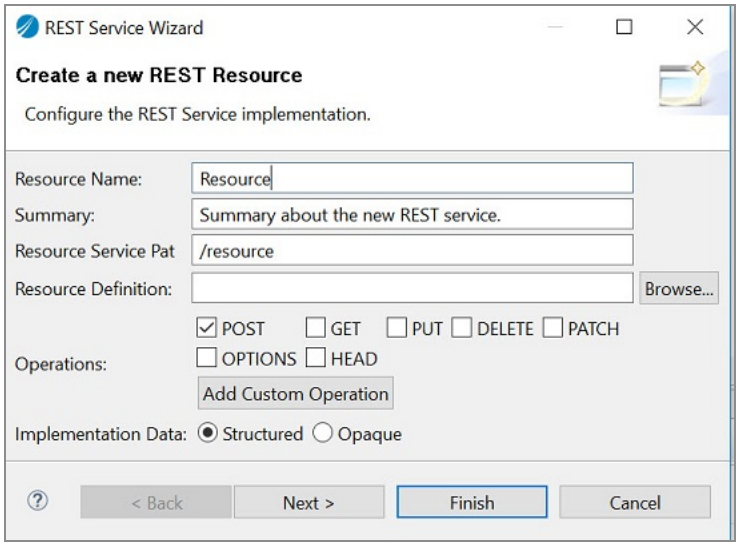
Procedure

1. In the **Project Explorer** view, 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.



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**.

REST Service Wizard



Create a new REST Resource

Configure the REST Service implementation.

Resource Name:	Resource
Summary:	Summary about the new REST service.
Resource Service Pat	/resource
Resource Definition:	
Operations:	[✓ POST] GET PUT DELETE PATCH OPTIONS HEAD
Implementation Data:	[✓ Structured] Opaque

REST Service Wizard fields

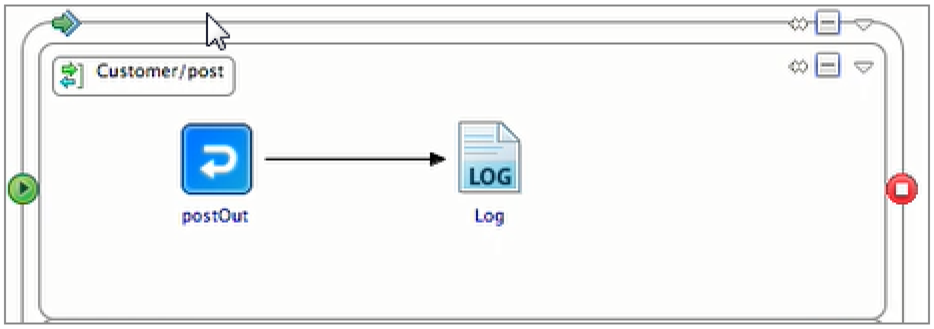
Field	Description
Resource Name	The name for the new REST service
Summary	Summary about the new REST service
Resource Service Path	Specifies the URI that is used to access the REST service
Resource Definition	Select a resource schema for the REST service, if needed
Operations	By default, the POST operation is selected. Select or deselect the operations as needed. **Note:** You can add custom operations by clicking on the **Add Custom Operation** button.
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 (the default nickname is of the format <*operation*><*resource_name*>), summary, and the request and response elements and their data types.
4. Click **Finish**.

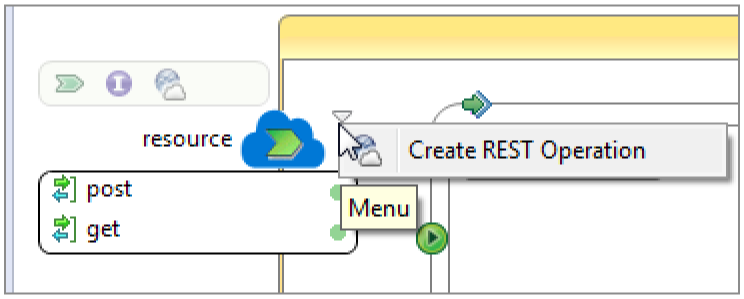
   The wizard adds the REST service and the selected operations, and also creates a process definition with the multiple operations. It generates a `.json` file in the **Service Descriptors** folder of your project when it creates the service.

   Note: 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 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 about the Swagger UI, see "Testing the REST Service" in *TIBCO BusinessWorks™ Container Edition Getting Started*.

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 for BusinessWorks.

The Rest Service Wizard has the following fields:

Field	Description
Resource Name	The name of the REST resource.
Summary	The summary or description of the REST resource.
Resource Service Path	The relative path for this REST service resource. If an application contains multiple REST bindings, ensure that the location of the path parameters is unique for each REST binding. 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 do not function correctly. In addition to path parameters, the path in a REST binding can also

Field	Description
	contain query parameters. For example, `/resource/path/{pathparam}? query={queryparam}` or `/resource/path/{pathparam}?{ queryparam}`
Resource Definition	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.
Operations	These are the HTTP methods implemented by this REST service. Currently POST, GET, PUT, PATCH, DELETE, OPTIONS, HEAD, and custom methods are supported for users to implement. **Note:** You can add custom operations by clicking on the **Add Custom Operation** button.
Implementation Data	The implementation data field can be **Structured** or **Opaque**. • **Structured**: 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. • **Opaque**: Use this mode to apply the pass through mechanism. You get a `messageBody` element in the input or output of every operation and then you must use either parse activities for JSON or XML to get a structured output for the payload.

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

Note: 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 has to be re-configured as the previous configurations are lost.

Using JSON Payload or an Existing XSD File

To use an existing JSON Payload or XSD file to create a service, import them into TIBCO Business Studio for BusinessWorks .

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. For more information, see Creating XML Schema From a JSON Payload . Then use the REST Service Wizard to create a service by following the instructions in Using the Wizard to Create a REST Service section.

REST Service Tutorial

The REST Bookstore sample lets you explore the REST tooling in TIBCO Business Studio for BusinessWorks. You can import this sample into TIBCO Business Studio for BusinessWorks through **File Explorer** and examine the project.

The processes in the sample implement different aspects of a bookstore, such as adding a book, deleting a book, and retrieving a list of books or a single book by its ISBN.

For more information about the sample, see "Using REST to Manage Books for a Bookstore" in *TIBCO BusinessWorks Container Edition Samples*.

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

Have the following installed or created on your computer:

- PostgreSQL
- Required database and tables
- Most recent version of Google Chrome web browser

For more information, see Installing PostgreSQL.

Installing PostgreSQL

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

Procedure

1. Download and install PostgreSQL from http://www.postgresql.org/download/

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

   Note: 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`. On Unix systems, use the first command in the readme to start the script from the **psql** window. On Windows, 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 to populate the database.

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

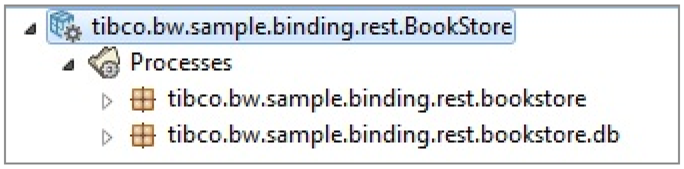
Creating a New Process

These steps show how to create a new process.

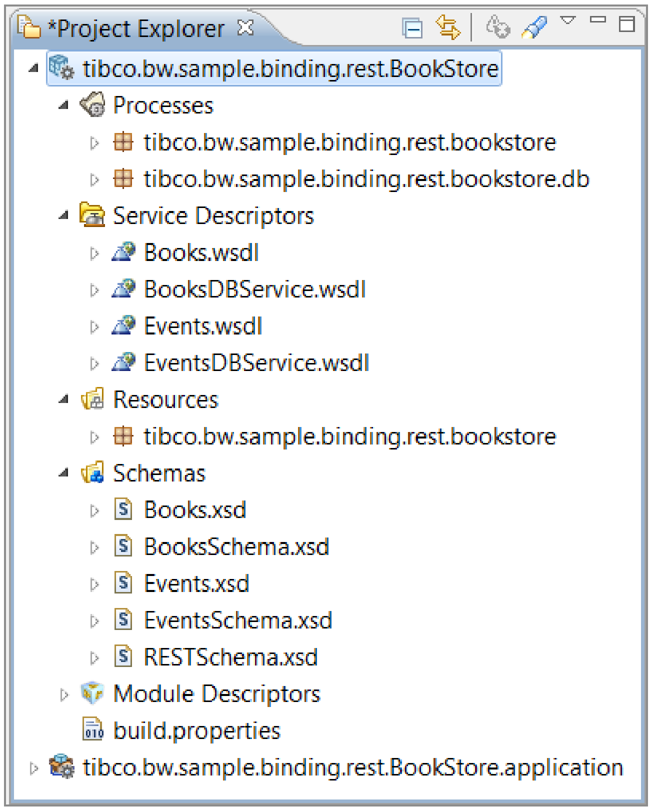
Procedure

1. Open TIBCO Business Studio for BusinessWorks.
2. Open the **Design** perspective by clicking Design 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

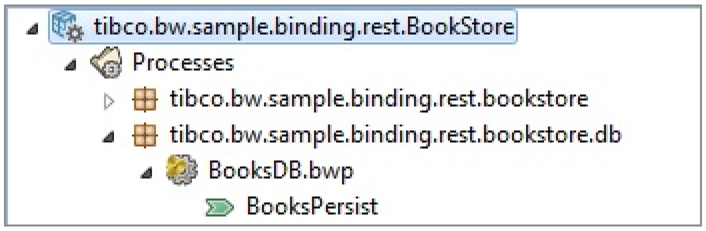
9. Expand the folders in the project to see all the project processes and resources. For more information on folder structure, see *TIBCO BusinessWorks™ Container Edition Application Development*.



10. Expand **Processes** and then expand **tibco.bw.sample.binding.rest.bookstore.db**.

    See **BooksDB.bwp**.

    > **Note:** The `.bwp` extension indicates that it is an TIBCO BusinessWorks Container Edition process.


tibco.bw.sample.binding.rest.BookStore
Processes
tibco.bw.sample.binding.rest.bookstore
tibco.bw.sample.binding.rest.bookstore.db
BooksDB.bwp
BooksPersist

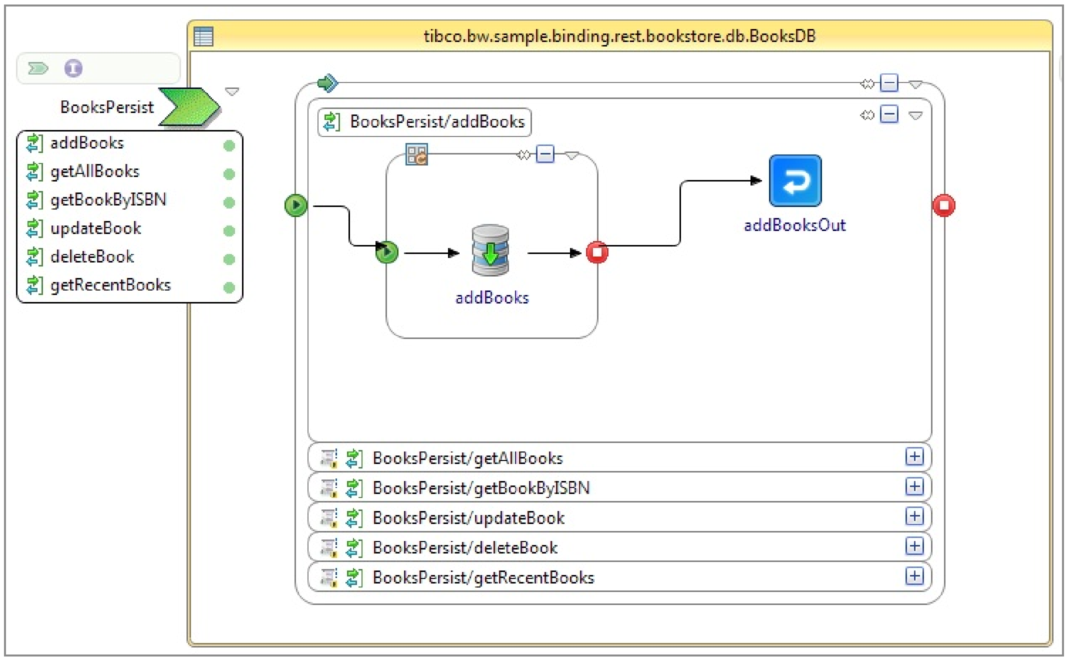
11. Double-click **BooksDB.bwp**.

    The 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.



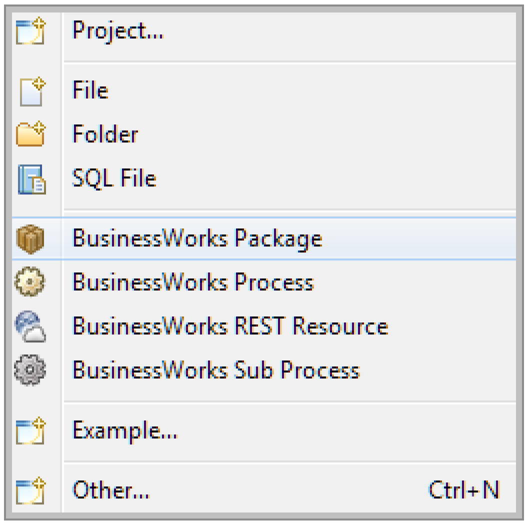
12. Double-click an operation to display the process for example, **BooksPersist > addBooks**.

    a. In the addBooks operation, you can see a JDBC activity.

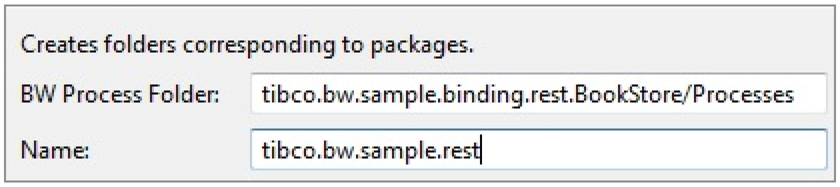
    b. The activity is repeated using a ForEach group.

    c. addBooksOut represents the **Reponse** 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** view, and select **New > BusinessWorks Package**.



14. In the BusinessWorks Package screen, specify tibco.bw.sample.rest in the **Name** field.

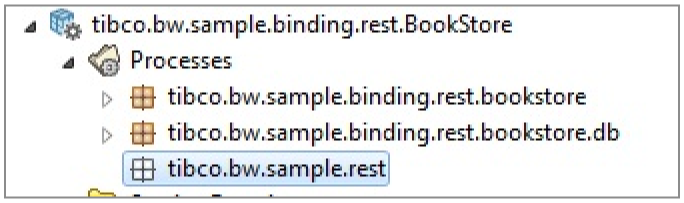


Creates folders corresponding to packages.

BW Process Folder: tibco.bw.sample.binding.rest.BookStore/Processes

Name: tibco.bw.sample.rest

15. Click **Finish** and verify that the new package `tibco.bw.sample.rest` has been added in the **Project Explorer** view.


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

Building a REST Service

This section details how to build a REST service.

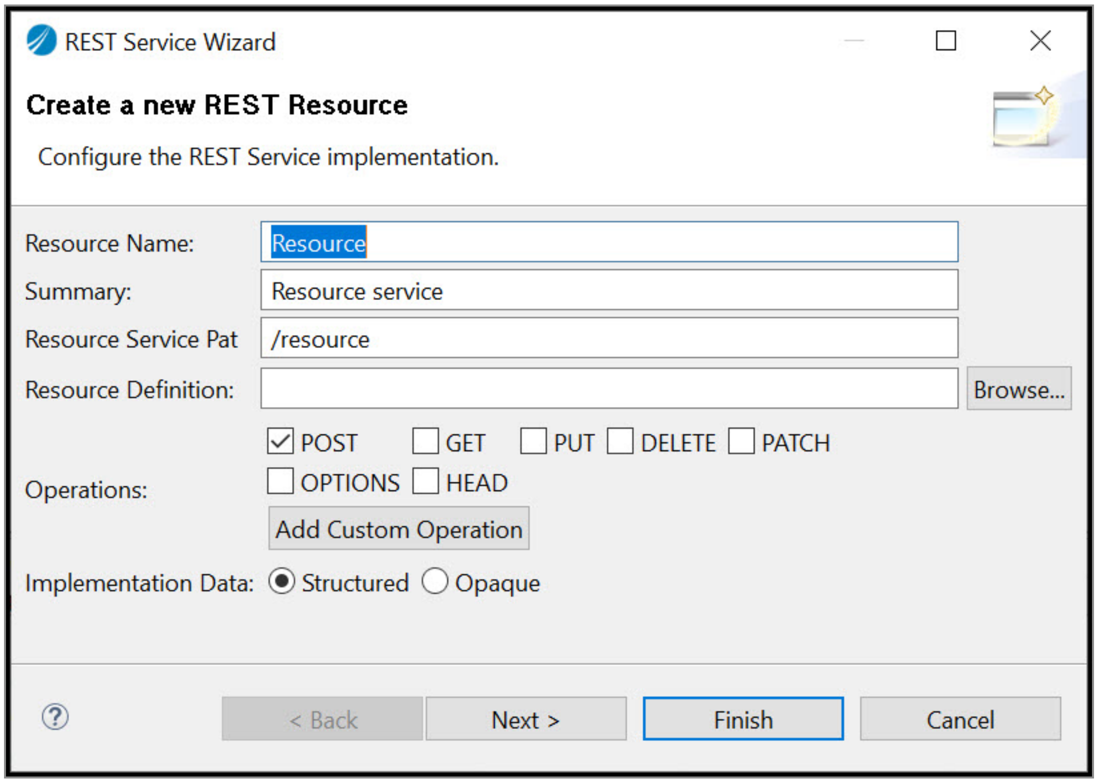
Before you begin

The **tibco.bw.sample.binding.rest.BookStore** sample is loaded in the Project Explorer.

Procedure

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

The REST Service Wizard window is displayed.



Create a new REST Resource

Configure the REST Service implementation.

Resource Name: Resource

Summary: Resource service

Resource Service Pat /resource

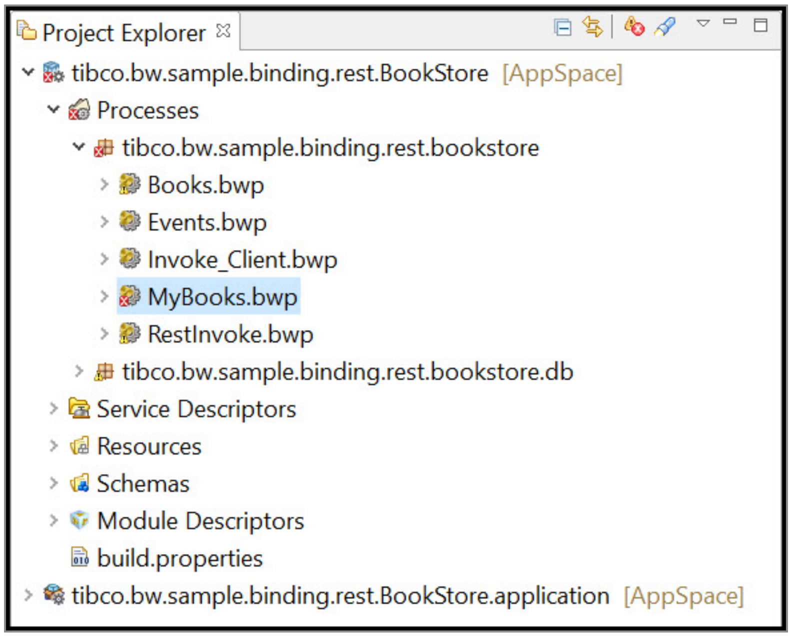
Resource Definition:

Operations: [✓ POST] GET PUT DELETE PATCH OPTIONS HEAD

Implementation Data: [✓ Structured] Opaque

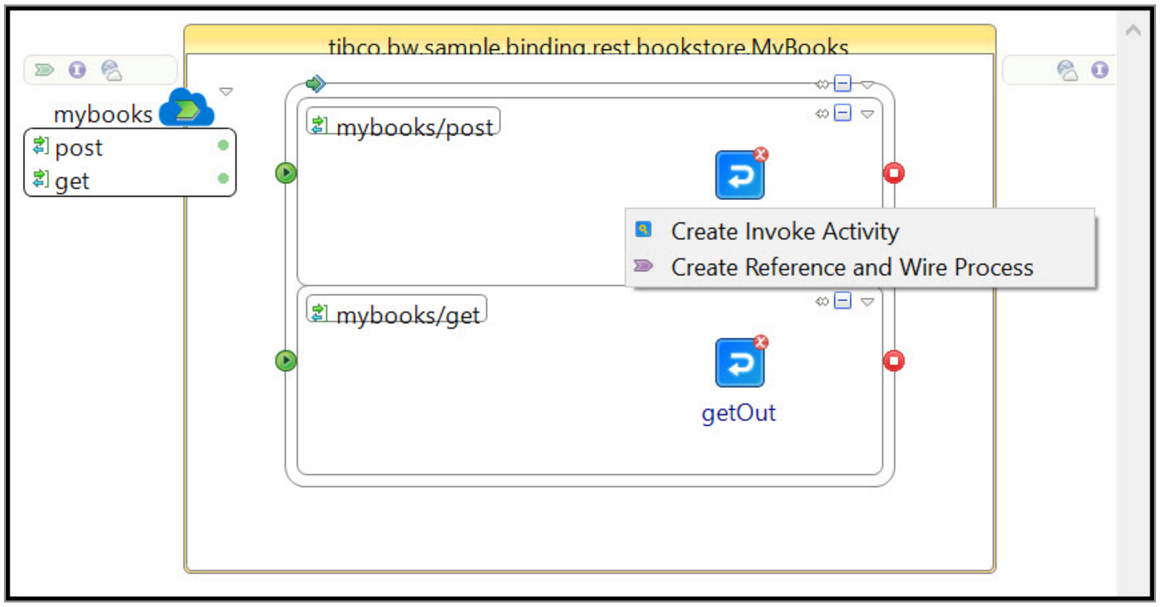
2. Specify the following values in the REST Service Wizard window.
   a. **Resource Name**: MyBooks
   b. **Summary**: Summary about the new REST service. (default)
   c. **Resource Service Path**: Auto-filled
   d. **Resource Definition**: Select **Browse > Schemas > Books.xsd > Books** in the Select 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**.


Project Explorer
tibco.bw.sample.binding.rest.BookStore [AppSpace]
Processes
tibco.bw.sample.binding.rest.bookstore
Books.bwp
Events.bwp
Invoke_Client.bwp
MyBooks.bwp
RestInvoke.bwp
tibco.bw.sample.binding.rest.bookstore.db
Service Descriptors
Resources
Schemas
Module Descriptors
build.properties
tibco.bw.sample.binding.rest.BookStore.application [AppSpace]

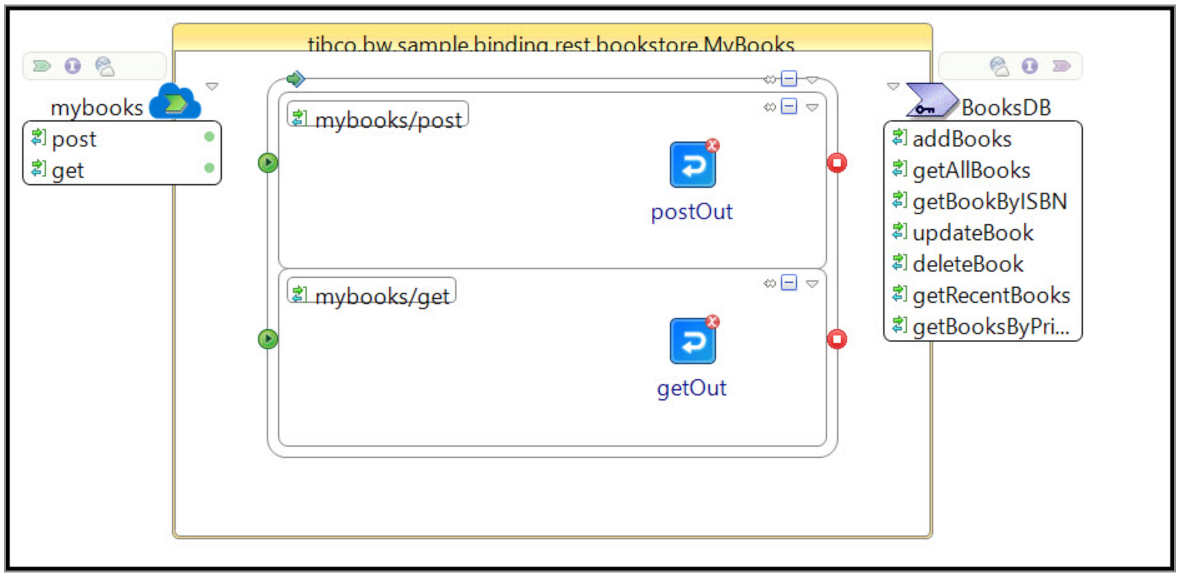
4. Open the **tibco.bw.sample.binding.rest.bookstore.db** package in the **Project Explorer** and select the **BooksDB.bwp** process. Drag it to the **Process Editor** and drop it on the implemented POST operation.

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


tibco.bw.sample.binding.rest.bookstore.MyBooks
mybooks
post
get
mybooks/post
Create Invoke Activity
Create Reference and Wire Process
mybooks/get
getOut

5. Select **Create References and Wire Process.**

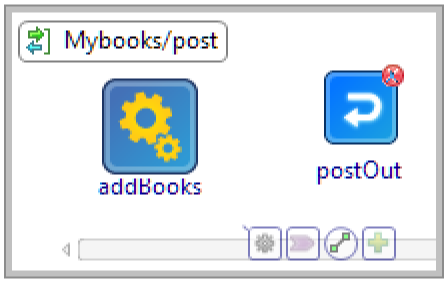
The references are added to the process. The purple chevron indicates the service and its operations that can be referenced by the process.



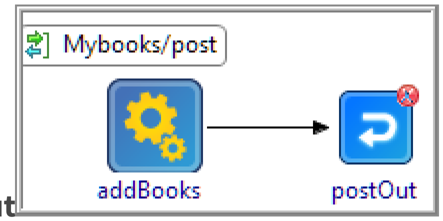
6. To update the POST process to invoke the appropriate external service operation:

   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.

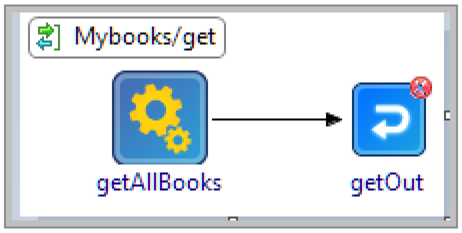


7. Click the newly added activity. Select the icon and connect **addBooks** to **postOut**



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

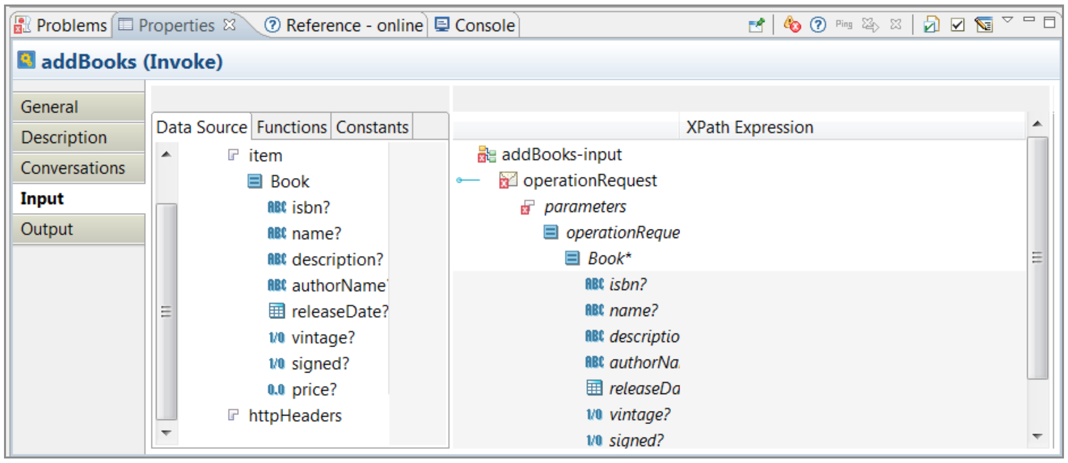
9. Connect **getAllBooks** to **getOut**.



10. Save your changes.

11. Click the **addBooks** activity and select **Properties > Input**.

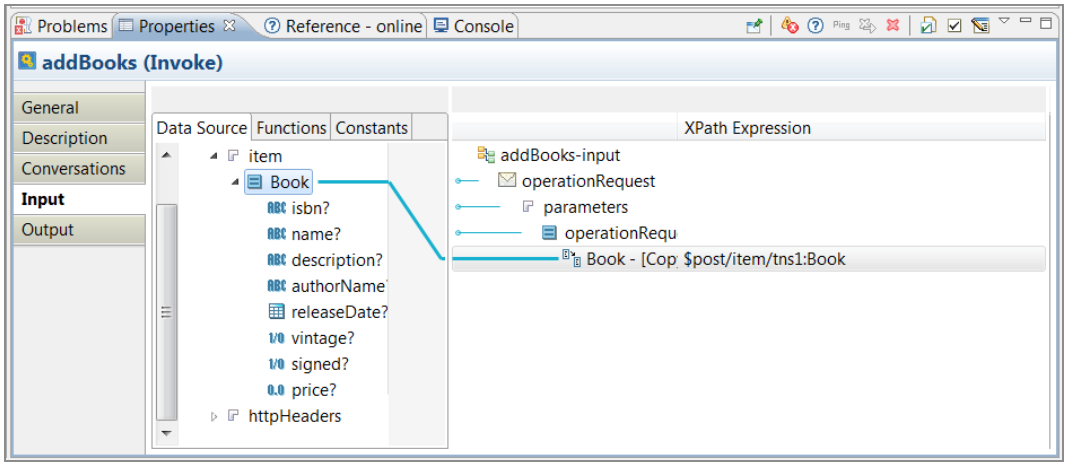
12. Expand the data tree in the **Data Source** pane to locate the Book element.



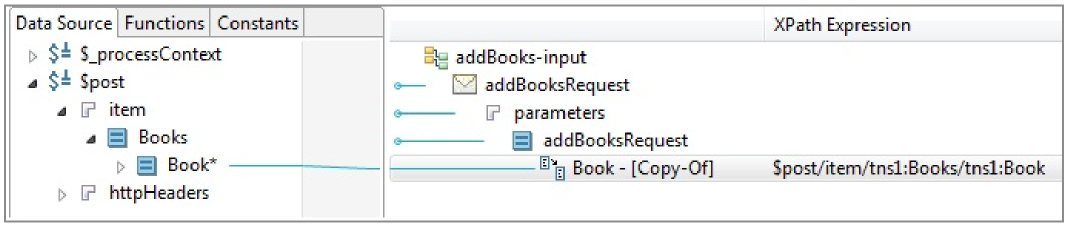
13. Drag the Book element from the left to the Book* element on the right.

14. In the pop-up window, select **Make a Copy of each " Book"** and click **Finish**.

    The **Input** tab looks like this:

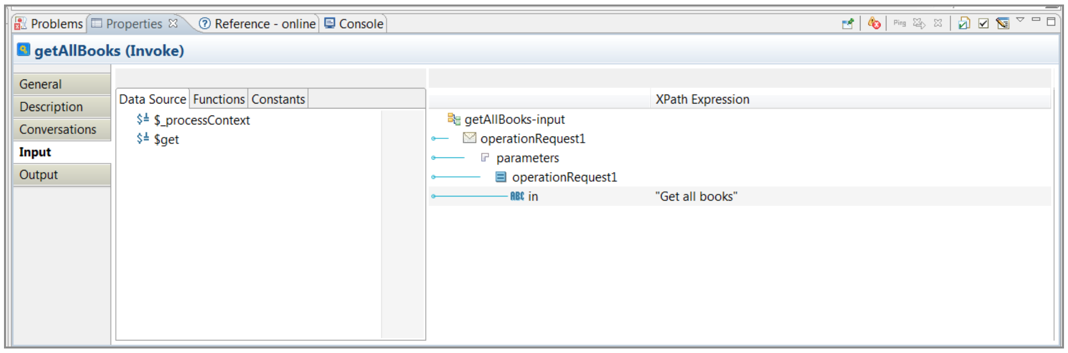


15. Save your changes.
16. Click the **postOut** activity and open the **Properties > Input** tab. Expand the **post** activity and drag the Book* element from left to right.
17. In the pop-up window, select the **For each** option and click **Next**. Click **Finish** on the **Auto-Map** window. The **Properties > Input** tab looks similar to this:



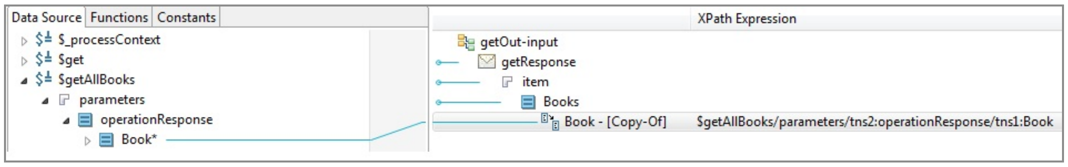
$post/item/tns1:Books/tns1:Book

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.



"Get all books"

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 looks similar to this:



$getAllBooks/parameters/tns2:operationResponse/tns1:Book

Result

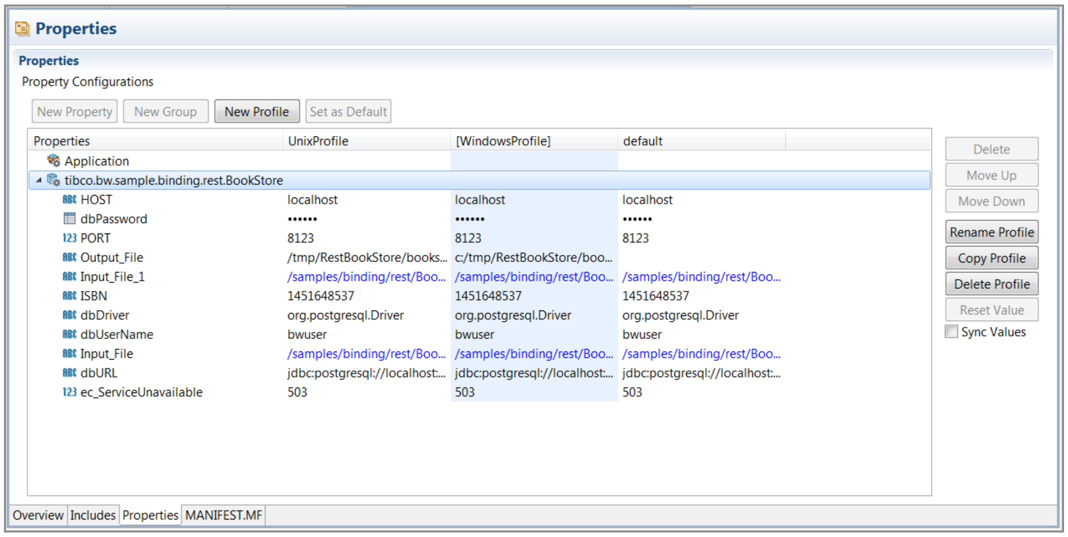
Your project is complete without any errors.

Testing the REST Service

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

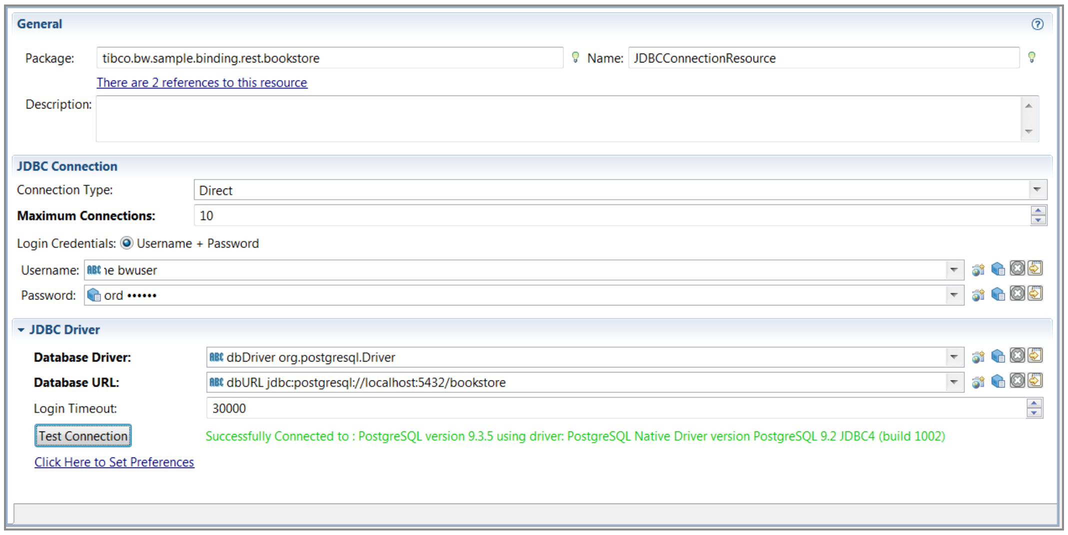
Procedure

1. In the **Project Explorer**, expand the 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	UnixProfile	[WindowsProfile]	default
Application			
tibco.bw.sample.binding.rest.BookStore			
HOST	localhost	localhost	localhost
dbPassword	••••••	••••••	••••••
PORT	8123	8123	8123
Output_File	/tmp/RestBookStore/books...	c:/tmp/RestBookStore/boo...	
Input_File_1	/samples/binding/rest/Boo...	/samples/binding/rest/Boo...	/samples/binding/rest/Boo...
ISBN	1451648537	1451648537	1451648537
dbDriver	org.postgresql.Driver	org.postgresql.Driver	org.postgresql.Driver
dbUserName	bwuser	bwuser	bwuser
Input_File	/samples/binding/rest/Boo...	/samples/binding/rest/Boo...	/samples/binding/rest/Boo...
dbURL	jdbc:postgresql://localhost:...	jdbc:postgresql://localhost:...	jdbc:postgresql://localhost:...
ec_ServiceUnavailable	503	503	503

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.


General

Package: tibco.bw.sample.binding.rest.bookstore
Name: JDBCConnectionResource
There are 2 references to this resource
Description:

JDBC Connection
Connection Type: Direct
Maximum Connections: 10
Login Credentials: Username + Password
Username: ıe bwuser
Password: ord ••••••

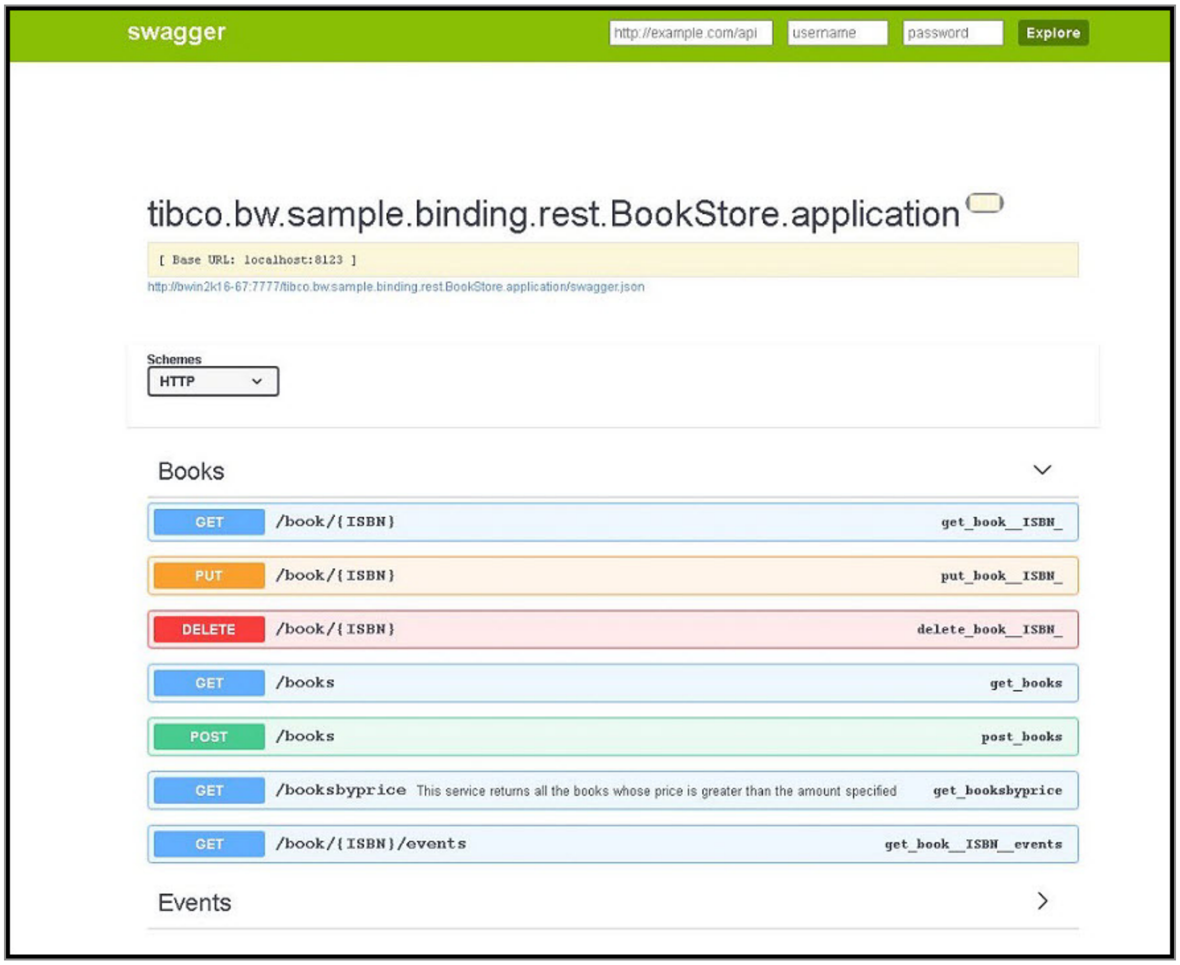
JDBC Driver
Database Driver: dbDriver org.postgresql.Driver
Database URL: dbURL jdbc:postgresql://localhost:5432/bookstore
Login Timeout: 30000
Test Connection
Successfully Connected to : PostgreSQL version 9.3.5 using driver: PostgreSQL Native Driver version PostgreSQL 9.2 JDBC4 (build 1002)
Click Here to Set Preferences

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**.
8. In the left-hand tree of the **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
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: [Application Name]: tibco.bw.sample.binding.rest.BookStore.application [Discovery Url]: http://localhost:7777/tibco.bw.sample.binding.rest.BookStore.application
12. Launch the Google Chrome browser.
13. Open http://localhost:7777/tibco.bw.sample.binding.rest.BookStore.application
14. Click **Books** or **Events** to see the operations. Click **MyBooks** to see the REST service

operations you just added. For more information, see the section Testing the Post and GET Operations.



tibco.bw.sample.binding.rest.BookStore.application

[Base URL: localhost:8123]

http://bwin2k16-67:7777/tibco.bw.sample.binding.rest.BookStore.application/swagger.json

Schemes: HTTP

Books

GET	/book/{ISBN}	get_book__ISBN_
PUT	/book/{ISBN}	put_book__ISBN_
DELETE	/book/{ISBN}	delete_book__ISBN_
GET	/books	get_books
POST	/books	post_books
GET	/booksbyprice This service returns all the books whose price is greater than the amount specified	get_booksbyprice
GET	/book/{ISBN}/events	get_book__ISBN__events

Events

15. 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:

```
{
  "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:

```
  {
      "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:

```
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
thirller","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":5
0}]}**************************************************************

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
thirller","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":5
0}]}**************************************************************

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":2
1}**************************************************************

DELETE Book By ISBN-------->"Deleted book with ISBN -
145164853
7"**************************************************************

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

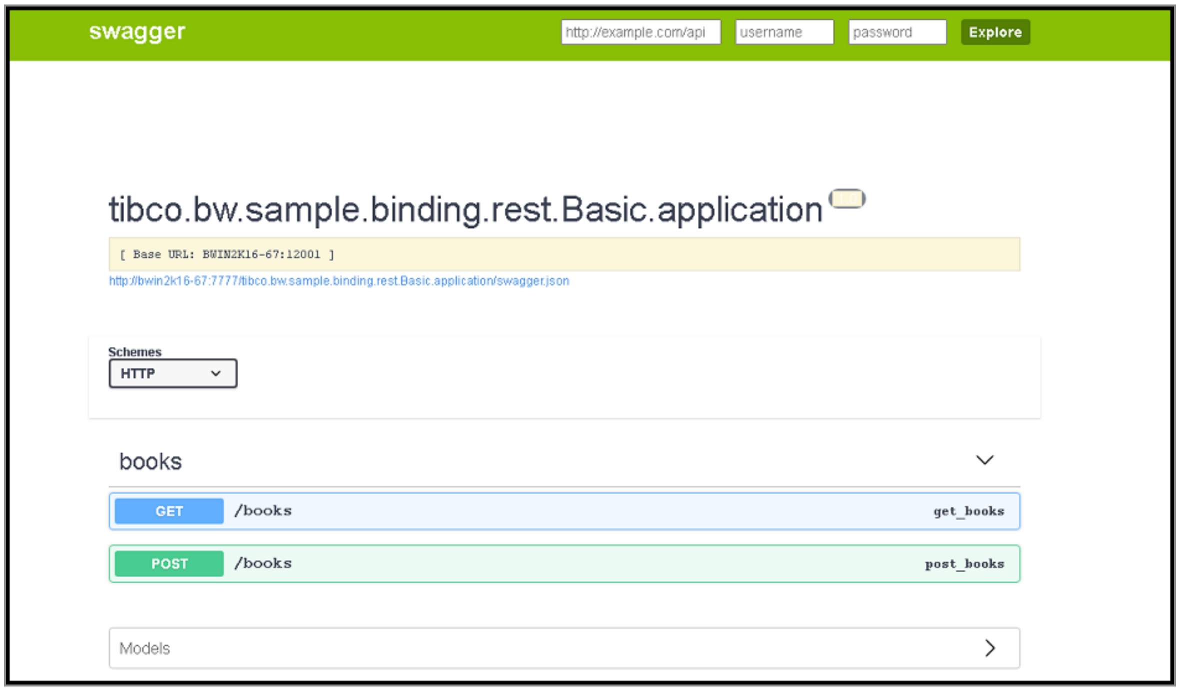
Testing the POST and GET Operations

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 **books**.

   It expands and displays the POST and GET operations.



tibco.bw.sample.binding.rest.Basic.application

[Base URL: BWIN2K16-67:12001]

http://bwin2k16-67:7777/tibco.bw.sample.binding.rest.Basic.application/swagger.json

Schemes: HTTP

books

GET /books get_books

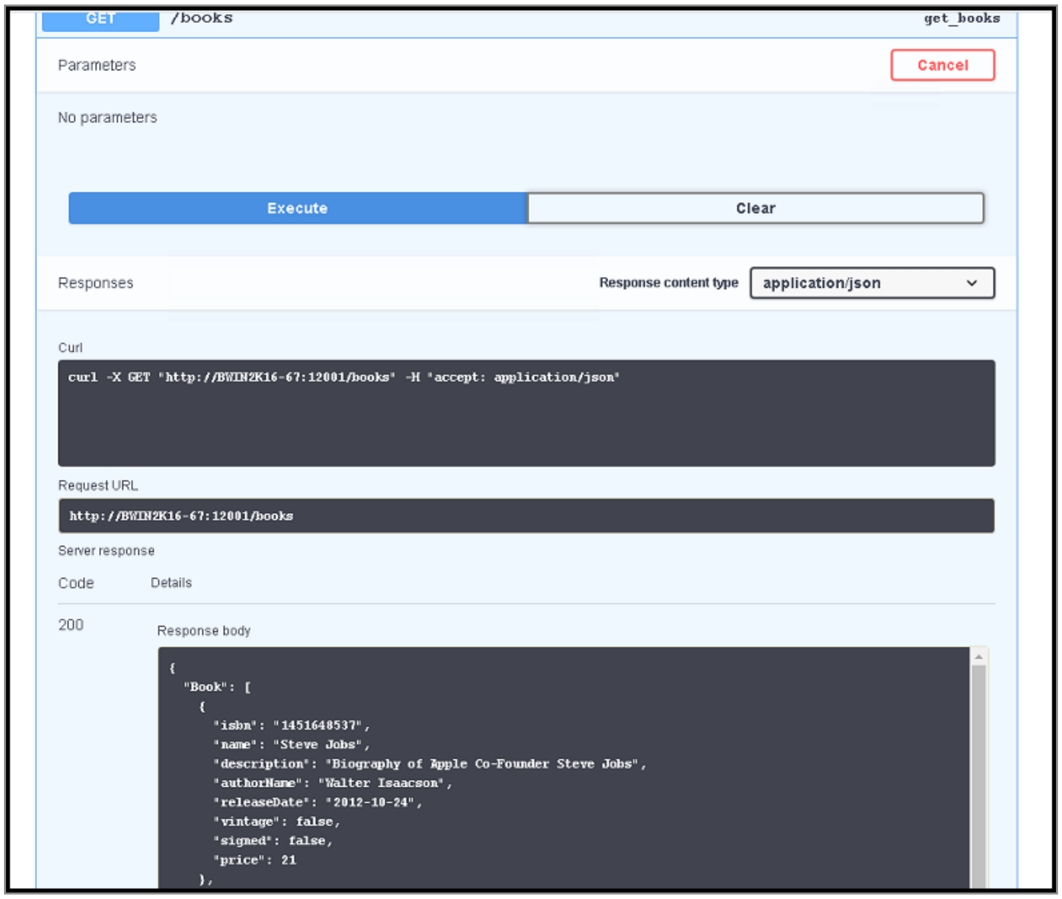
POST /books post_books

Models

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

3. Click the **Try it out!** button.
4. Provide values to the Books parameter and then click **Execute** button. You can use the JSON payload in the `<BW_HOME>\samples\binding\rest\BookStore\samplejson` folder.
5. Now click the **GET** icon to display its details.
6. Click the **Try it out!** button.
7. Click the **Execute** button.

   The response displays a list of books returned by the REST service from the database.



```
curl -X GET "http://BWIN2K16-67:12001/books" -H "accept: application/json"
```

```
http://BWIN2K16-67:12001/books
```

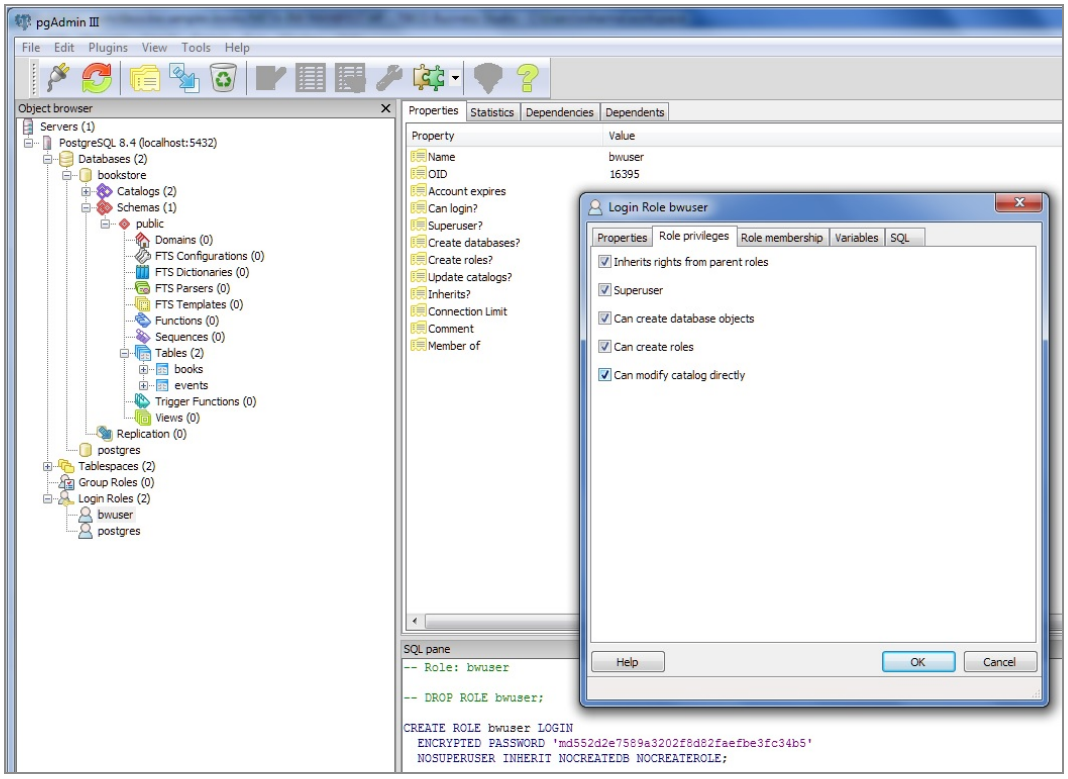
```
{
  "Book": [
    {
      "isbn": "1451648537",
      "name": "Steve Jobs",
      "description": "Biography of Apple Co-Founder Steve Jobs",
      "authorName": "Walter Isaacson",
      "releaseDate": "2012-10-24",
      "vintage": false,
      "signed": false,
      "price": 21
    },
```

8. After you have finished, go back to TIBCO Business Studio™ for BusinessWorks™ and click 🟥 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.

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



Property	Value
Name	bwuser
OID	16395
Account expires	
Can login?	
Superuser?	
Create databases?	
Create roles?	
Update catalogs?	
Inherits?	
Connection Limit	
Comment	
Member of	

- [✓] Inherits rights from parent roles
- [✓] Superuser
- [✓] Can create database objects
- [✓] Can create roles
- [✓] Can modify catalog directly

```
-- Role: bwuser

-- DROP ROLE bwuser;

CREATE ROLE bwuser LOGIN
  ENCRYPTED PASSWORD 'md552d2e7589a3202f8d82faefbe3fc34b5'
  NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE;
```

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 POST, GET, PUT, PATCH, and DELETE HTTP methods. It also supports JSON, XML, and plain text message types.

Binding

This section has the following fields.

Field	Description
Implementation Resource or Resource Service Path	The Swagger specification in the project that 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.
Resource Service Path	Resource path of the REST Resource to invoke. **Note:** Path parameters that are not immediately enclosed in forward slashes are supported. For example, the parameter `authorName('{isbn}')` in the resource service path `/book/authorName('{isbn}')/` is not directly contained by forward slashes, but passes successfully.
HTTP Client	The name of the HTTP Client. **Tip:** Click on the **HTTP Client Name** field to display details about the HTTP Connector resource.

Field	Description
Request Client Format	The type of request message format. The two available reply message format options are: JSON or XML. Applies to all operations unless overridden at the operation level.
Response Client Format	The type of reply message format. The two available reply message format options are: JSON or XML. Applies to all operations unless overridden at the operation level.
Path Parameters	If Path parameters are defined in the REST service that the REST Reference is calling, they appear in the Path Parameters table. However, you cannot modify the Path parameters since the REST Reference must adhere to the parameters defined by the service.

Summary tab

This tab shows the following details.

Field	Description
Summary	A brief description of the operation.
HTTP Method	A unique identifier for the operation that identifies the operation in the entire API. By default, it is set to *<HTTP-Method>-<resource_name>.*
Notes	A field that can be used to describe the operation. Any text that is entered in the Notes field appears in the Swagger file.

Request tab

This tab shows the following details.

Field	Description
Use Empty Values for Null	Select the check box to set empty values instead of NULL values in JSON. i.e use [] brackets instead of NULL.

Field	Description
	By default, the check box is clear.
Format	Supported formats for REST service request are: • JSON • XML • Text
Request Entity Processing	This field has two values: • **BUFFERED**: the request entity is buffered in memory to determine the content length that is sent as a Content-Length header in the request. • **CHUNKED**: the entity is sent as chunked encoded (no Content- Length is specified, entity is streamed). The Transfer-Encoding header is set to Chunked. The default value is Chunked.
JSON Definition Style	Specifies whether the request item is a Single Element or an Anonymous Array.
Request	Data type of the Payload. It can be one of the following: • XSD Element • String • Integer • Boolean • Form Data - Tag/Value (application/x-www-form-urlencoded) • Form Data - Multipart (application/form-data)
Query and Header Parameters	The user can perform following operations: • Add Query Parameter • Add Header Parameter

Field	Description
	• Remove Parameter • Scroll Up • Scroll Down This pane has four columns: • Parameter Name Name of the parameter. Users can edit the parameter name by clicking on the parameter added. **Note:** Do *not* use a space in the HTTP header name. If you do, the following error is displayed: `Bad message- 400 - Illegal character.` • Type Data type of the parameter. It can be: ◦ String ◦ Integer ◦ Long ◦ Float ◦ Double ◦ Boolean ◦ Byte ◦ Binary ◦ Date ◦ Date Time ◦ Password • Repeating This field can be toggled to Yes and No. • Required This field can be toggled to Yes and No.

Response tab

This tab shows the following details.

Field	Description
Use HTTP Headers	Select this check box only if you require a service to send back a custom header, or value or if you need to see the response headers. This check box is selected by default if custom headers are used. When you select this check box, you can add custom HTTP fault headers defined in the **Response Status** tab.
Use Null for Empty values	Select the check box to set NULL values instead of empty values in JSON. i.e use NULL values instead of [] brackets. By default, the check box is clear.
Ignore Additional JSON Fields	Select this check box to ignore additional fields that are generated due to changes in the external payload when processing the schema. By default, the check box is clear.
Format	Response format requested by the client, can be **JSON**, **XML** or **Text**. The service must support the formats that the client requests for this operation.
JSON Definition Style	Specifies whether the request item is a **Single Element** or an **Anonymous Array**.
Resource Schema	These are the available options: • String • Integer • Boolean • XSD element: 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.

Field	Description
	Note: Make sure the schema resource you select does not contain cyclic dependencies on other schemas , or a type that has two child members with the same local name, but different namespaces.
Header Parameters	This field is enabled only when you select the **Use HTTP Headers** check box. The user can perform following operations: • Add Header Parameter • Remove Parameter • Scroll Up • Scroll Down This pane has four columns: • Parameter Name Name of the parameter. Users can edit the parameter name by clicking on the parameter added. **Note:** Do *not* use a space in the HTTP header name. If you do, the following error is displayed:`Bad message- 400 - Illegal character.` • Type Data type of the parameter. It can be: ◦ String ◦ Integer ◦ Long ◦ Float ◦ Double ◦ Boolean ◦ Byte ◦ Binary

Field	Description
	◦ Date ◦ Date Time ◦ Password • Repeating This field can be toggled to Yes and No. • Required This field can be toggled to Yes and No.

Response Status tab

This tab shows the following details.

Field	Description
Code	These are unique numbers. Click on the error code to customize it. **Note:** Use custom status code 200 only when the response is not defined, that is, when the **Response with status code only** check box is selected in the **Response** tab.
Type	Data type of the error code. Following types are supported: • 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. • String • Integer • Boolean The default type is String.
Reason Phrase	Description of the error code. Click on the value to customize the description.

Enabling JSON Payload Logging

To enable raw JSON payload logging for both Inbound and Outbound of the **Invoke** activity for REST binding in debug logging, add the following VM arguments:

- -Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog
- -Dorg.apache.commons.logging.simplelog.showdatetime=true
- -Dorg.apache.commons.logging.simplelog.log.org.apache.http=DEBUG
- -Dorg.apache.commons.logging.simplelog.log.org.apache.http.wire=DEBUG

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 TIBCO Business Studio for BusinessWorks. You can only view the details of such a reference. References that are created using the REST Reference Wizard can be modified in TIBCO Business Studio for BusinessWorks because these 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 Swagger to Create a REST Reference

If you have the Swagger file, you can import it into TIBCO Business Studio for BusinessWorks 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 TIBCO Business Studio for BusinessWorks. You can only view the details of such a reference.

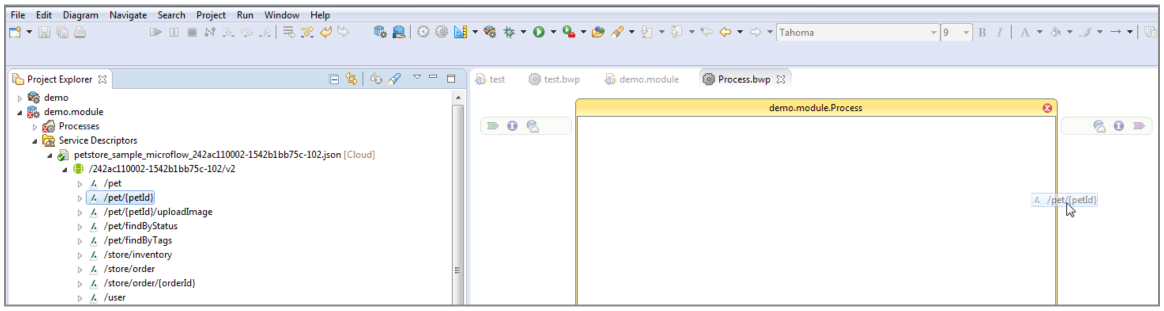
Before you begin

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. For Swagger files that reside on the cloud, you can use the **API Explorer** view to get the Swagger specifications that were created in API Modeler. Drag and drop the Swagger file into the **Service Descriptors** folder.

For more information, see the API Explorer topic.

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

In TIBCO Business Studio for BusinessWorks, you can 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 for BusinessWorks 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.
In the wizard, you can choose the operation(s) to implement in the reference and 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.

Before you begin

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

Note: When invoking external REST services using the **REST Reference Wizard**, the Request and Response of the REST method must be configured with the corresponding XSD schema element used by the invoked service, otherwise the service is not invoked.

To create a REST reference, follow the steps:

Obtain the URL for the service endpoint

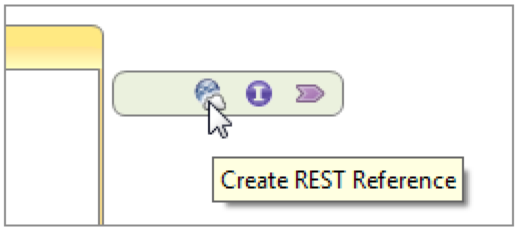
Typically, the URL for an endpoint can be obtained from the API web page or some other documentation where the service is described. This documentation should list the details of the service such as operations that the service supports, the parameters used and the data definitions for the requests and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

Verify the service is running

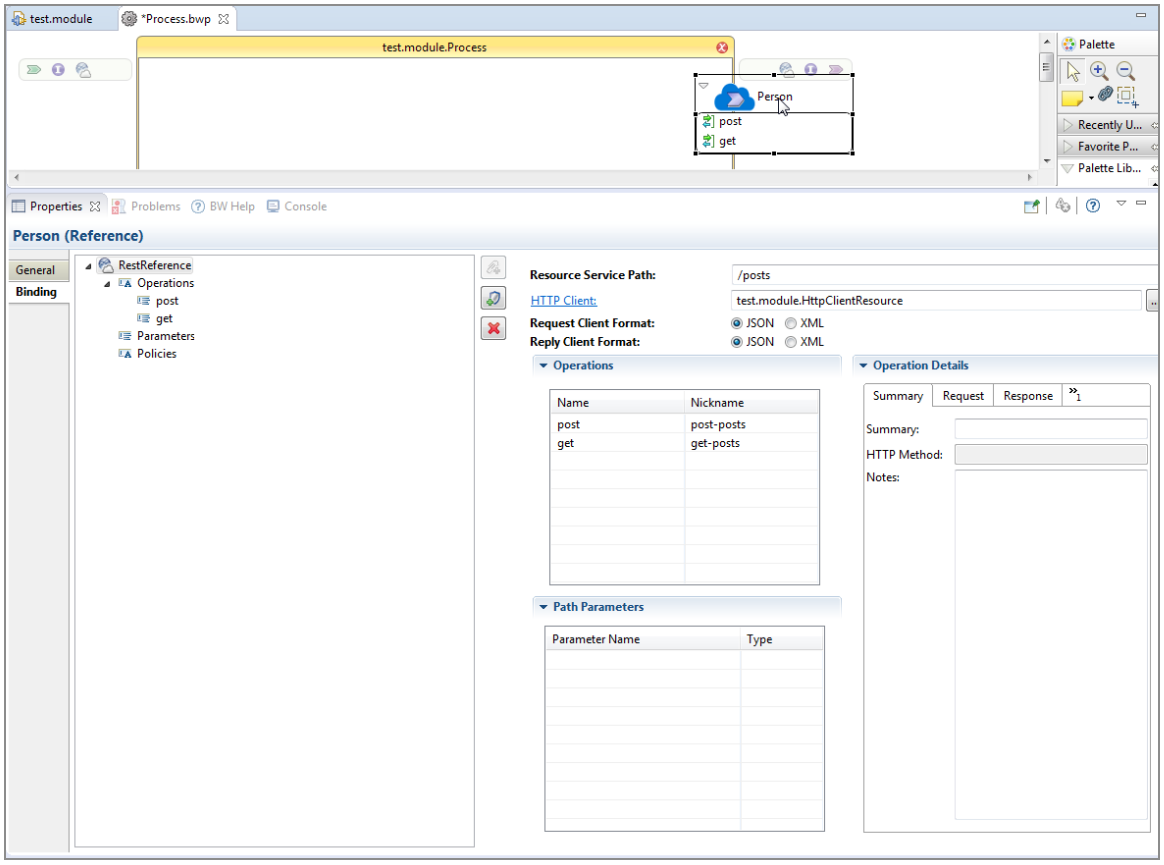
You can verify that the service is running by invoking it using a testing tool such as Postman or SoapUI.

Create a new application and open the REST Reference Wizard

1. Create a new application if you do not already have one. For more information, see "Creating an Application" in *TIBCO BusinessWorks™ Container Edition Application Development*.
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 selecting respective check boxes 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 on the extreme right of your Process Editor.
6. Open the Reference properties view by clicking on the reference name.



Resource Service Path: /posts
HTTP Client: test.module.HttpClientResource
Request Client Format: [✓ JSON] XML
Reply Client Format: [✓ JSON] XML

Name	Nickname
post	post-posts
get	get-posts

Parameter Name	Type

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.

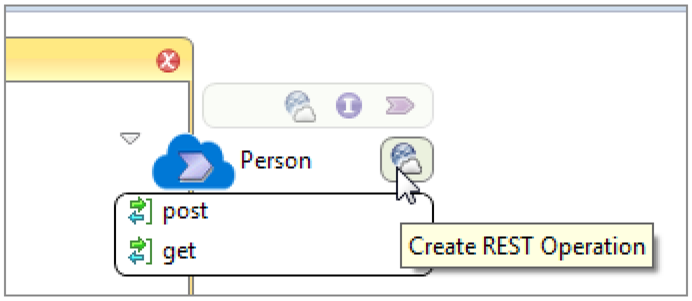
Note: Do not use port 7777. It is reserved for internal use.

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.

     For more information about how to add parameters, see Parameter Support section.

- 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:



  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 have the option to configure your request and response for the operation at a later time too from the Reference properties view.

You can now invoke any operation by dragging and dropping it in the process editor.

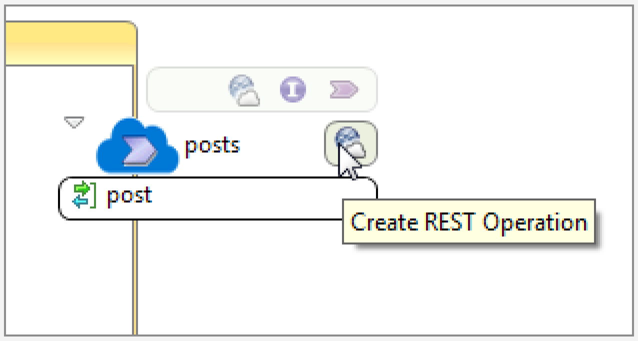
REST Reference Wizard

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

The Rest Reference Wizard has the following fields:

Field	Description
Service URL	The URL to the REST Service.
Operations	These are the HTTP methods implemented by the REST Reference process to access the REST service. Currently only POST, GET, PUT, PATCH, and DELETE methods are supported.

Note: To add additional operations to an existing REST reference process, click **Create Rest Operation**. This icon appears when you hover over the reference in the Process Editor as follows:



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.

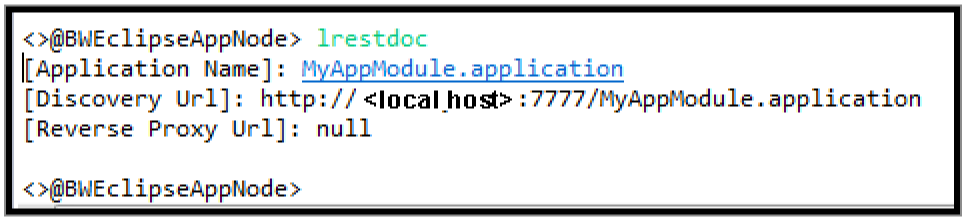
If you make schema changes to the APIs that were created ground up by you in TIBCO Business Studio for BusinessWorks, the Swagger for such APIs automatically gets updated by TIBCO Business Studio for BusinessWorks .

Using the Debug Engine

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

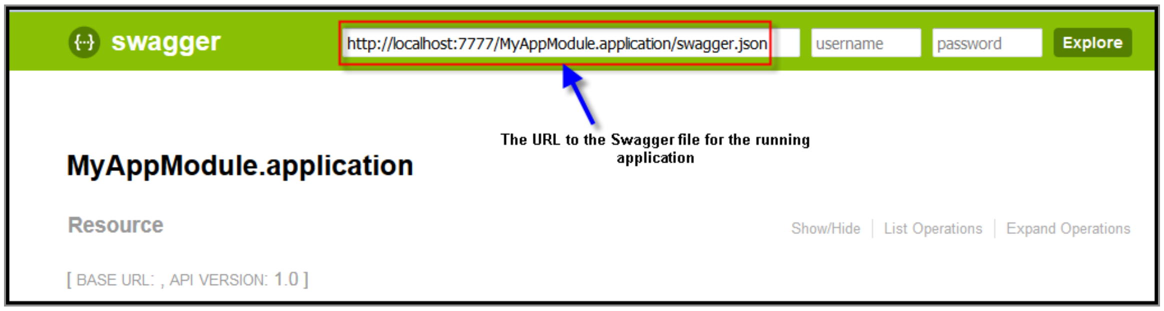
Procedure

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.



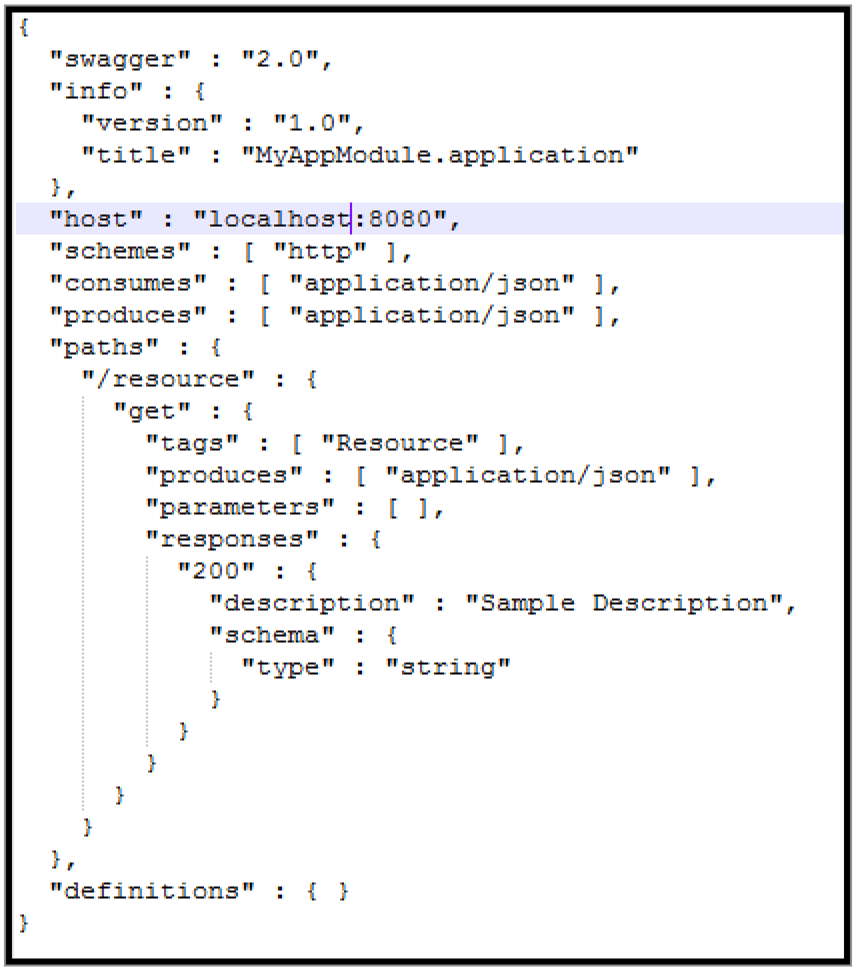
```
<>@BWEclipseAppNode> lrestdoc
[Application Name]: MyAppModule.application
[Discovery Url]: http://<localhost>:7777/MyAppModule.application
[Reverse Proxy Url]: null

<>@BWEclipseAppNode>
```

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 displays the path to the Swagger file of the running service.



swagger
http://localhost:7777/MyAppModule.application/swagger.json
username
password
Explore
The URL to the Swagger file for the running application
MyAppModule.application
Resource
Show/Hide
List Operations
Expand Operations
[BASE URL: , API VERSION: 1.0]

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.



```
{
  "swagger" : "2.0",
  "info" : {
    "version" : "1.0",
    "title" : "MyAppModule.application"
  },
  "host" : "localhost:8080",
  "schemes" : [ "http" ],
  "consumes" : [ "application/json" ],
  "produces" : [ "application/json" ],
  "paths" : {
    "/resource" : {
      "get" : {
        "tags" : [ "Resource" ],
        "produces" : [ "application/json" ],
        "parameters" : [ ],
        "responses" : {
          "200" : {
            "description" : "Sample Description",
            "schema" : {
              "type" : "string"
            }
          }
        }
      }
    }
  },
  "definitions" : { }
}
```

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 contains all the information such as the hostname or the machine where the service is running, the port number and the base path for the service. The reference you create reaches 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. 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 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 port allows that property value to be provided by an administrator that is running the application. Change the value 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 .

   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. To see the newly created module property, expand the application module tree.
8. Click the up arrow button to promote the module property to application property.

   Now the IP address is 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.

Before you begin

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 Business Studio for BusinessWorks.
2. Open the **Design** perspective by clicking the **Design** icon in the upper right corner.
3. Click **File > New > Other > BusinessWorks > BusinessWorks Application Module** and click **Next**.
4. Enter tibco.bw.sample.binding.rest in the **Project Name** text box. Do not change the remaining default settings.
5. Click **Finish**. This step creates a new application module with an empty process.
6. Obtain the Swagger file from the Swagger UI of the running service.
7. Copy and paste the content into 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.

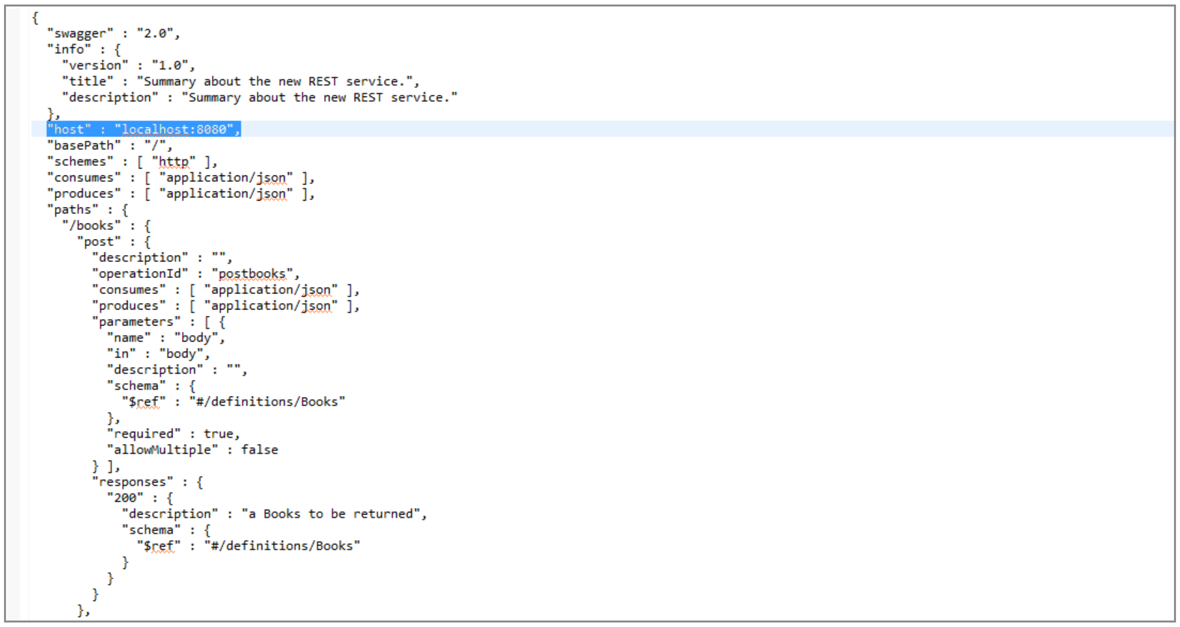
2. Right-click **Service Descriptors** and select **Import > Import... > General > File System** and click **Next**.
3. In the File system dialog box, click the **Browse** button and browse to the location of the `Books.json` file.
4. Select the check box next to **Books.json** in the left pane and click **Finish**.

Creating the REST Reference

1. In the **Project Explorer**, completely expand the **tibco.bw.sample.binding.rest** folder under **Service Descriptors**.
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 **IntervalInMillisec** 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.
10. Select the check box under **Fix** and click **OK**.
11. 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.
12. Expand **$get** in the **Data Source** tab completely.
13. In the XPath Expression pane, expand the **post-input** completely.
14. Drag and drop **Book*** from the **Data Source** tab to the **Book*** under post-input in the

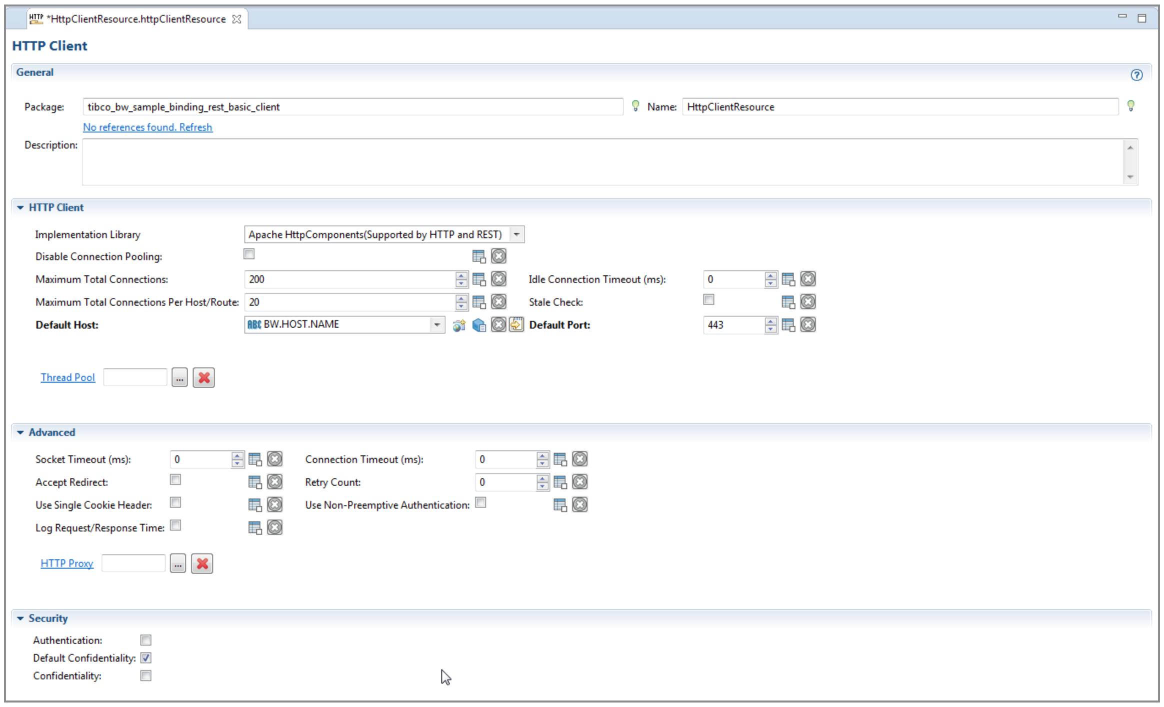
XPath Expression pane.

15. In the Drop dialog, select **Make a copy of each "book"** radio button and click **Finish**.
16. Click **Show Check and Repair** icon in the icon bar on the upper right corner of the Properties view.
17. Select the check box under **Fix** and click **OK**.
18. Click **Show Check and Repair** icon again. Select the check box under **Fix** and click **OK**.
19. 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.



```
{
  "swagger" : "2.0",
  "info" : {
    "version" : "1.0",
    "title" : "Summary about the new REST service.",
    "description" : "Summary about the new REST service."
  },
  "host" : "localhost:8080",
  "basePath" : "/",
  "schemes" : [ "http" ],
  "consumes" : [ "application/json" ],
  "produces" : [ "application/json" ],
  "paths" : {
    "/books" : {
      "post" : {
        "description" : "",
        "operationId" : "postbooks",
        "consumes" : [ "application/json" ],
        "produces" : [ "application/json" ],
        "parameters" : [ {
          "name" : "body",
          "in" : "body",
          "description" : "",
          "schema" : {
            "$ref" : "#/definitions/Books"
          },
          "required" : true,
          "allowMultiple" : false
        } ],
        "responses" : {
          "200" : {
            "description" : "a Books to be returned",
            "schema" : {
              "$ref" : "#/definitions/Books"
            }
          }
        }
      },
```

20. Expand the Resources folder under the **tibco.bw.sample.binding.rest.basic** application module.
21. Double-click **HttpClientResource.httpClientResource**.
22. In the HTTP Client section, change the Default Host to `BW.HOST.NAME` and Default Port to `443` and select the **Default Confidentiality** check box. This is required when accessing services running in the cloud.



*HttpClientResource.httpClientResource

HTTP Client

General

Package: tibco_bw_sample_binding_rest_basic_client
Name: HttpClientResource
No references found. Refresh
Description:

HTTP Client

Implementation Library: Apache HttpComponents(Supported by HTTP and REST)
Disable Connection Pooling: ☐
Maximum Total Connections: 200
Idle Connection Timeout (ms): 0
Maximum Total Connections Per Host/Route: 20
Stale Check: ☐
Default Host: BW.HOST.NAME
Default Port: 443
Thread Pool

Advanced

Socket Timeout (ms): 0
Connection Timeout (ms): 0
Accept Redirect: ☐
Retry Count: 0
Use Single Cookie Header: ☐
Use Non-Preemptive Authentication: ☐
Log Request/Response Time: ☐
HTTP Proxy

Security

Authentication: ☐
Default Confidentiality: ✓
Confidentiality: ☐

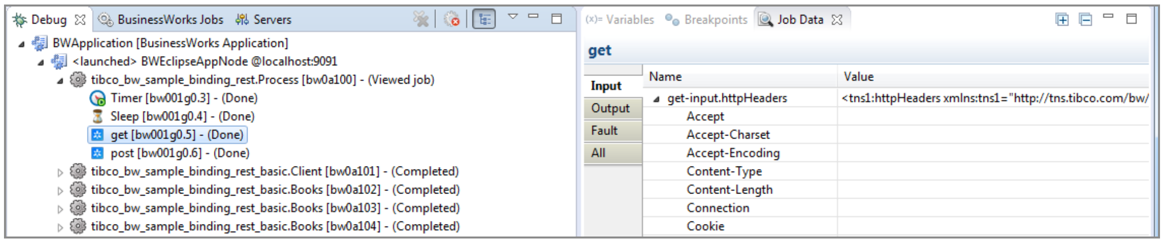
23. Click **File > Save All**.

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, then click **Deselect All** if you have multiple applications.
4. Select the check boxes next to **tibco.bw.sample.binding.rest.basic.application** and **tibco.bw.sample.binding.restapp**.
5. 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]
6. In the **Debug** view, expand **BWApplication [BusinessWorks Application] > <launched> BWEclipseAppNode > tibco.bw.sample.binding.rest.Process** and select **get**.

7. In the **JobData** view, you can see the job data of the **get** activity.



BWApplication [BusinessWorks Application]
<launched> BWEclipseAppNode @localhost:9091
tibco_bw_sample_binding_rest.Process [bw0a100] - (Viewed job)
Timer [bw001g0.3] - (Done)
Sleep [bw001g0.4] - (Done)
get [bw001g0.5] - (Done)
post [bw001g0.6] - (Done)
tibco_bw_sample_binding_rest_basic.Client [bw0a101] - (Completed)
tibco_bw_sample_binding_rest_basic.Books [bw0a102] - (Completed)
tibco_bw_sample_binding_rest_basic.Books [bw0a103] - (Completed)
tibco_bw_sample_binding_rest_basic.Books [bw0a104] - (Completed)

get

Input | Output | Fault | All

Name	Value
get-input.httpHeaders	<tns1:httpHeaders xmlns:tns1="http://tns.tibco.com/bw/
Accept	
Accept-Charset	
Accept-Encoding	
Content-Type	
Content-Length	
Connection	
Cookie	

Refactoring a REST Service or Reference

You can change the configuration of your REST service even after it is created in TIBCO Business Studio for BusinessWorks.
The steps to 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 TIBCO Business Studio for BusinessWorks.

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 is updated by its provider, update your implementation of that file to get the changes.

The icon to the left side of the Swagger file in the Project Explorer in the TIBCO Business Studio for BusinessWorks 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 it source. You can compare the differences between the two and update your local copy. 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 appears in the **Synchronize** tab and if you double click it, 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 regenerates the schema. It updates it in both the service and reference.

   Note: Nothing changes for processes that have already been created.

Updating a Service Created within TIBCO Business Studio for BusinessWorks

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

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.

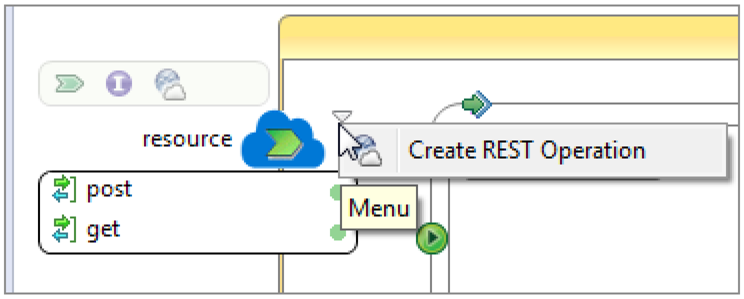
   The Summary page for that operation opens on the right. To edit the request, response or response status for the operation, click the respective tabs.

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**.



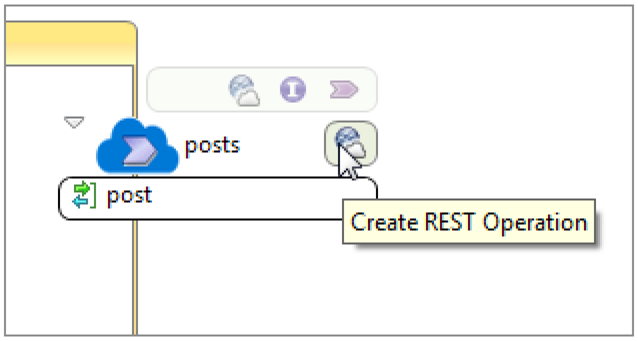
2. In the REST Service Wizard, select a schema for the operation using the **Browse** button for **Resource Definition**.
3. Select the operation check box.
4. Optionally, click **Next** to configure the operation. You can configure the operation after you have added it from the **Components** page. For more information, see Updating Configuration.
5. Click **Finish**.

Adding an Operation After the REST Reference is Created

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 the operation check box.
4. Optionally, click **Next** to configure the operation. You can configure the operation

after you have added it from the **Components** page. For more information, see Updating Configuration.

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
[Endpoint Type] : REST
[Endpoint URL] : http://localhost:8123
[CLIENT FORMAT ] : JSON
[RESOURCE PATH ] : /book/{ISBN}
[HTTP METHODS] : GET, PUT, DELETE
[Endpoint Type] : REST
[Endpoint URL] : http://localhost:8123
[CLIENT FORMAT ] : JSON
[RESOURCE PATH ] : /books
[HTTP METHODS] : POST, GET
[Endpoint Type] : REST
[Endpoint URL] : http://localhost:8123
[CLIENT FORMAT ] : JSON
[RESOURCE PATH ] : /book/{ISBN}/events
[HTTP METHODS] : GET
[Endpoint Type] : REST
[Endpoint URL] : http://localhost:8123
[CLIENT FORMAT ] : JSON
[RESOURCE PATH ] : /event/{EventID}
[HTTP METHODS] : GET, PUT, DELETE
[Endpoint Type] : REST
[Endpoint URL] : http://localhost:8123
[CLIENT FORMAT ] : JSON
[RESOURCE PATH ] : /events
[HTTP METHODS] : POST, GET
```

Exception Handling

Errors (or faults) can occur when executing a process. Using fault handlers you can 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, as a result you can 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 does 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 checks 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.

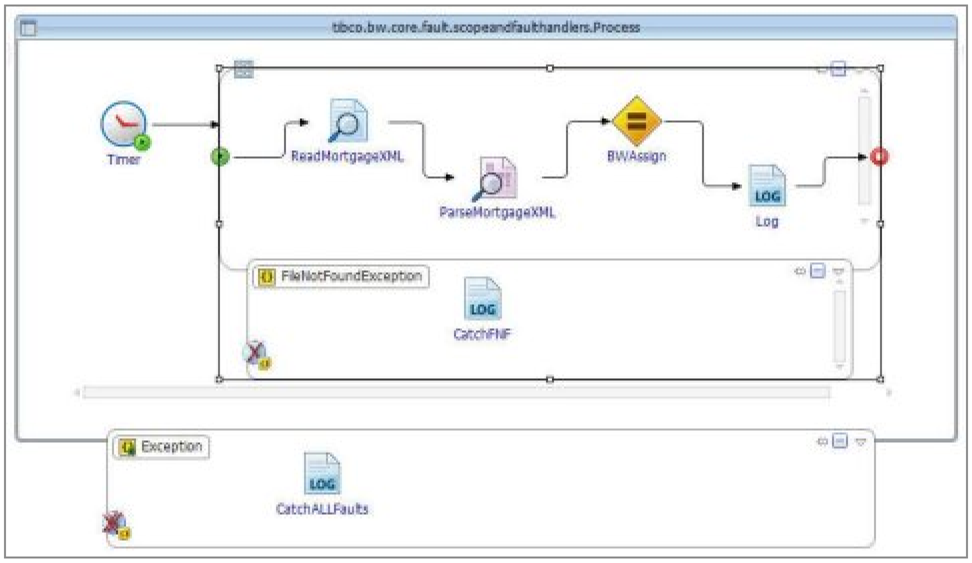
Creating Faults

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, as a result you can catch faults or exceptions thrown by activities within a scope. There are two types of fault handlers: **Catch Specific Fault** and **Catch All Faults**.

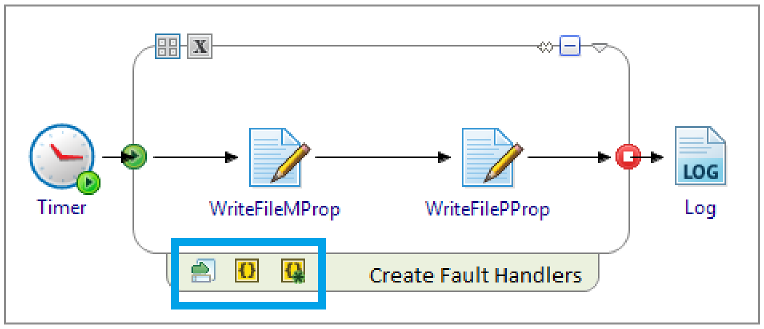
Fault handlers can be defined at the process level, or at a scope level within a process. The diagram below shows two fault handlers - one defined at the process level and the other defined at an inner scope level.

Fault Handler Attached to an Inner Scope



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.

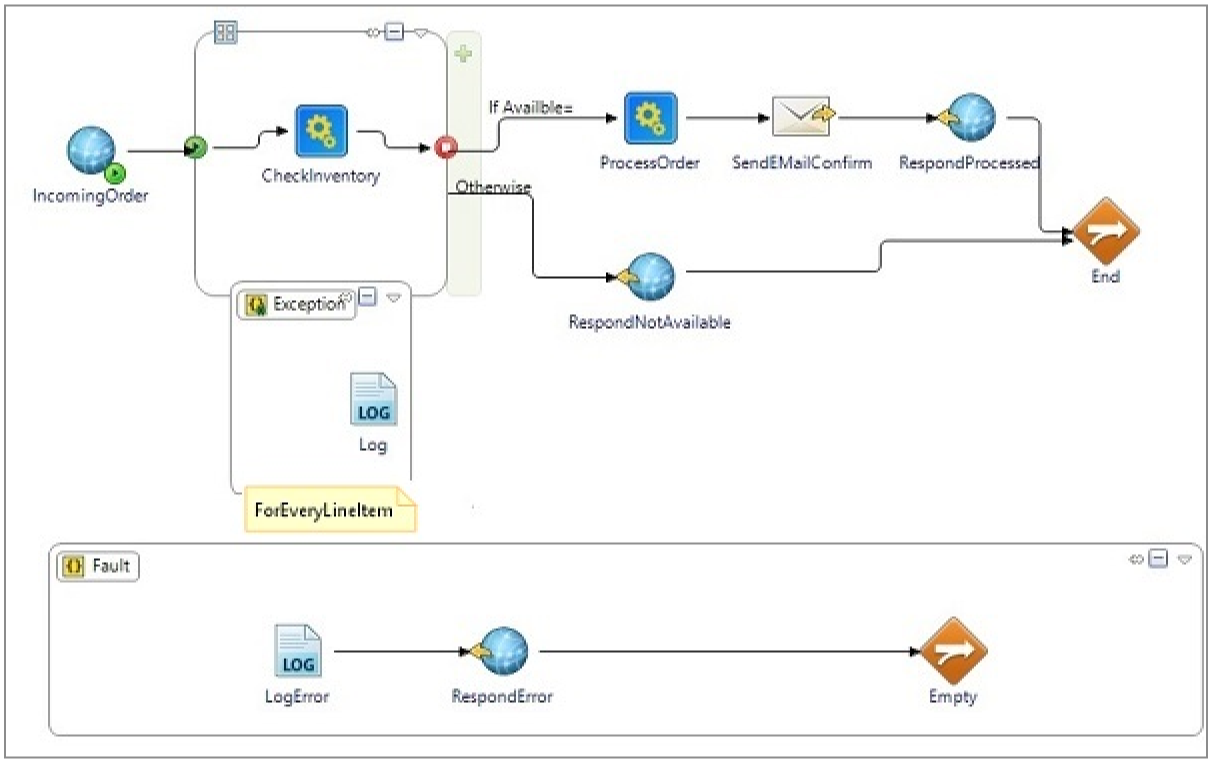
Using the Catch and Rethrow Activities

You can place a Catch block in your process to deal with unhandled exceptions. Using the Catch block, you can 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 after which the transition is taken to the end of the group. If the inventory is available, the order is processed, , 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. 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.

Using the **Rethrow** activity, you can 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.

For more information about the Catch , see the *TIBCO BusinessWorks Container Edition Bindings and Palettes Reference* guide.

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 for BusinessWorks 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. 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.

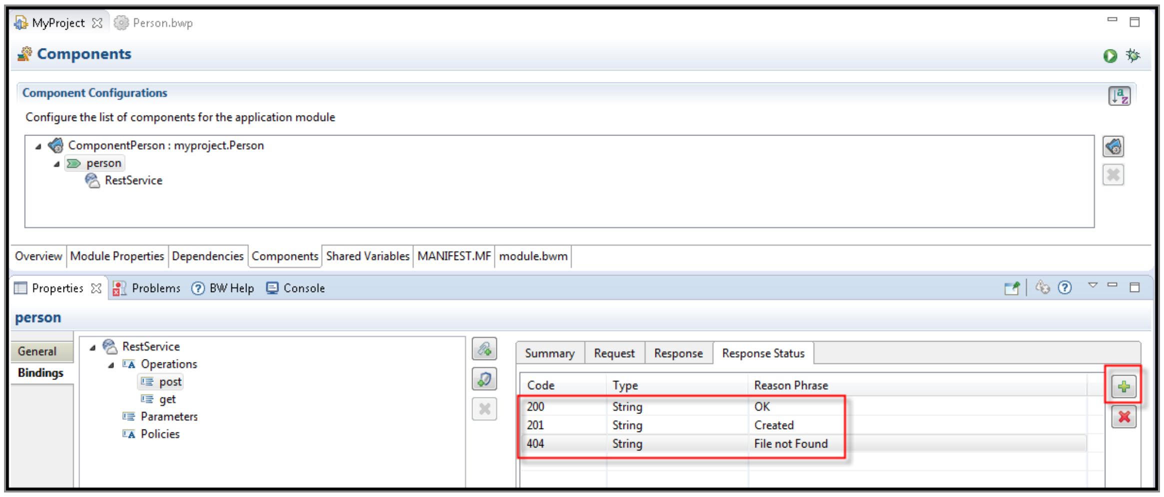
Note: You can edit error codes only in projects that were created ground up in TIBCO Business Studio for BusinessWorks . In projects that were created using an imported Swagger file, you can only view the error code details.

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 is shown by default in the default description for that error code. You can click on it to add more details to the error description.

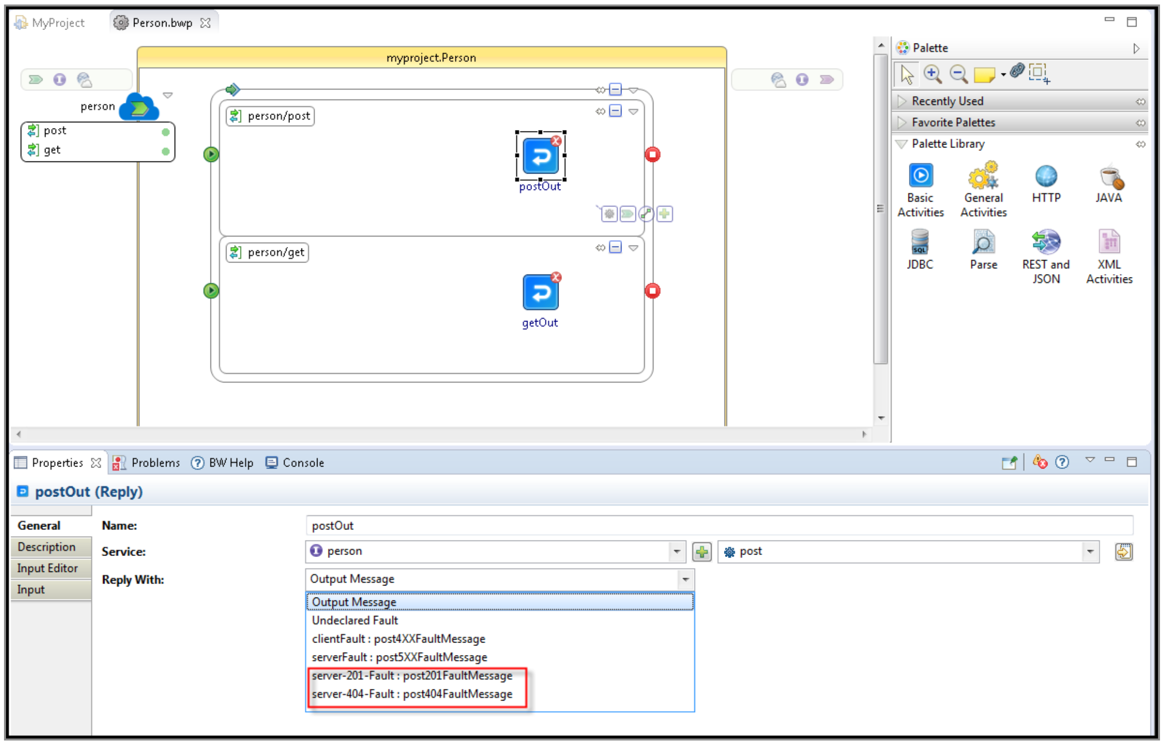


Code	Type	Reason Phrase
200	String	OK
201	String	Created
404	String	File not Found

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

Result

The newly added error code is available for you to select.



postOut (Reply)

Name:	postOut
Service:	person / post
Reply With:	Output Message

Output Message
Undeclared Fault
clientFault : post4XXFaultMessage
serverFault : post5XXFaultMessage
server-201-Fault : post201FaultMessage
server-404-Fault : post404FaultMessage

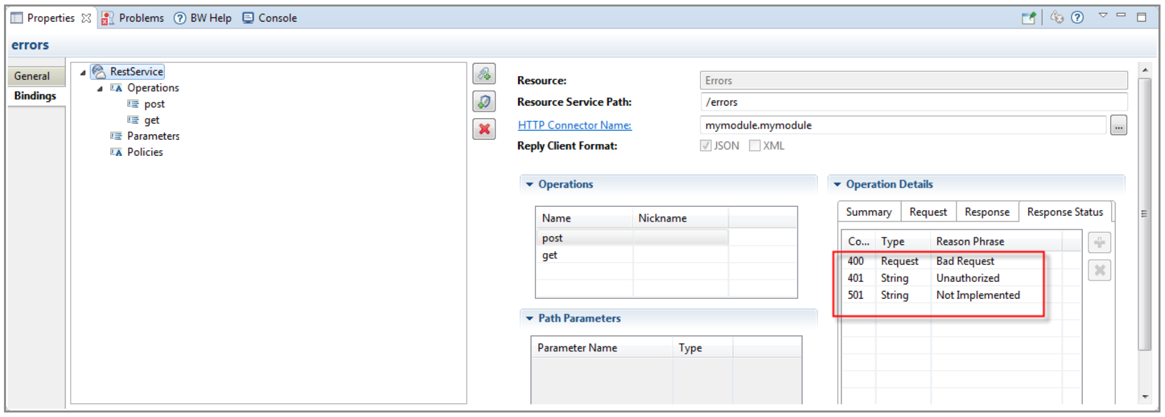
Using Swagger-Provided Faults

Fault codes that are defined in a Swagger file that is used in TIBCO Business Studio for BusinessWorks 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.



Resource: Errors
Resource Service Path: /errors
HTTP Connector Name: mymodule.mymodule
Reply Client Format: ✓ JSON ☐ XML

Operations

Name	Nickname
post	
get	

Path Parameters

Parameter Name	Type

Operation Details

Summary | Request | Response | Response Status

Co...	Type	Reason Phrase
400	Request	Bad Request
401	String	Unauthorized
501	String	Not Implemented

Standard HTTP Status Codes

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

The following are the most commonly used response codes:

Error Code	Description
200	OK
201	Created
202	Accepted
204	No Content
301	Moved Permanently
303	See Other
304	Not Modified
307	Temporary Redirect
400	Bad Request

Error Code	Description
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

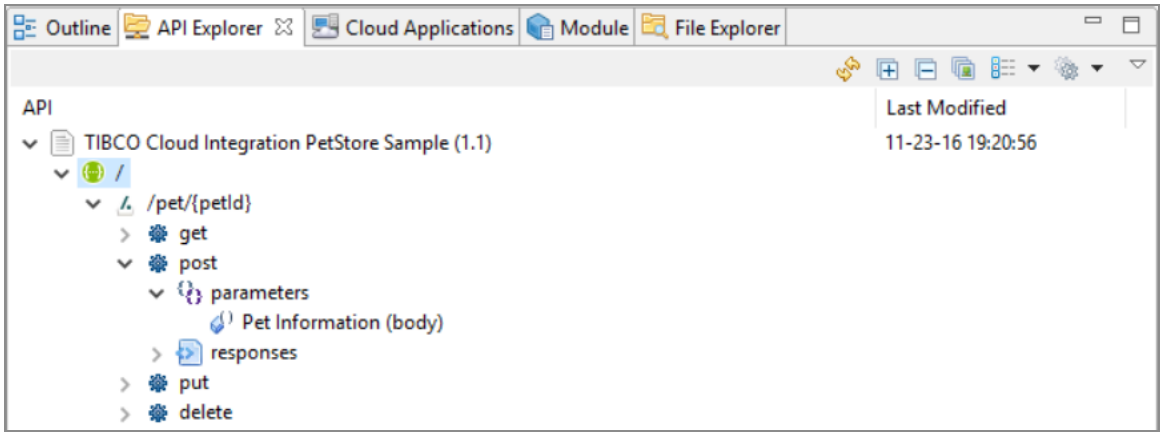
For more information, see Status Code and Reason Phrase at w3.org.

Eclipse Views for REST

TIBCO Business Studio for BusinessWorks has **API Explorer** view that is useful when working with REST APIs.

API Explorer

Displays a connected view of the TIBCO BusinessWorks Container Edition API Modeler residing in the cloud. This view shows abstract APIs that were created in API Modeler. You can also view the APIs residing on your local machine from the **API Explorer**.



```
API                                                        Last Modified
v TIBCO Cloud Integration PetStore Sample (1.1)            11-23-16 19:20:56
    v /
        v /pet/{petId}
            > get
            v post
                v parameters
                      Pet Information (body)
                > responses
            > put
            > delete
```

When you open TIBCO Business Studio for BusinessWorks for the very first time, enter your credentials for the registry site by opening the Settings dialog box and double-clicking on the registry name and entering your username and password for the site in the resulting dialog box. To open the Settings dialog, click the (▽) button on the upper right corner of the **API Explorer** view and click **Settings**. This populates the **API Explorer** view with the APIs that are available in the registry.

Adding a new registry to the API Explorer view

Use the Settings dialog in the **API Explorer** to add a new registry (location) from where you want to view the APIs. To open the Settings dialog box, click the (▽) button on the upper right corner of the **API Explorer** view, and click **Settings**.

By default, the Settings dialog box is configured with a Cloud registry which is set to the URL for the API Modeler.

To create a new registry:

1. Click the **New** button.
2. Enter a name for the registry **Name** field.
3. Select whether the registry is pointing to a local folder on your machine (**Local Folder**) or to a URL in the cloud (**Cloud**).
4. Provide the location of the registry in the **URL** field. If the registry points to a location on the cloud, you need to provide the authentication details for it in the **Username** and **Password** text boxes.
5. Click **Finish**.

To edit an existing registry entry:

1. Click the name of the registry and click **Edit**.
2. Make your edits to the entry. You can change the name of the registry, delete the registry configuration by clicking **Remove**, or changing the order in which the registries show up in the API Explorer by using the **Up** and **Down** button.
3. Click **Finish** when you are done with your edits.

Select a specific registry entry check box to display the registry in the **API Explorer** view. For more information, see Filtering the APIs in the API Explorer View.

Setting the presentation of the APIs in the API Explorer view

In this dialog box, you can specify how the discovered APIs appear in the **API Explorer** view:

- **API Presentation** - specifies how the APIs appear in the **API Explorer** view

  Flat - displays the APIs as a flat list with each API's version number displayed next to its name in parenthesis. If there are multiple versions of the same API, each version is shown as a separate API, hence multiple APIs with the same name but different version numbers.

  Hierarchical - displays every API as a hierarchy of API name label with version number folder under it and the actual API under the version folder. If there are multiple versions for an API, each version is listed in its own separate folder under the API name label.

Latest Version - displays only the latest version of the API, even though there might be multiple versions available.

- **Group by API registry** - groups the APIs according to the registry from which they were discovered. You also have the option to display the URL of the APIs next to the registry name by selecting the **Show API Registry URL** check box.
- **Group apps by sandbox** - If you have multiple sandboxes that contain apps, the **Cloud Applications** view displays the sandboxes and groups the apps under their respective sandbox.
- **Check Supported Plugins** - If your application uses plug-ins, you must verify that the plug-ins are supported in before you push the application to the cloud. You can do so by clicking this button.

You should now see the APIs displayed in the API Explorer in the format that you specified in the **Settings** dialog. Expanding an API shows you its version, the resource path, and the operations you can perform on that resource.

The **API Explorer** view has the following quick-access buttons that you can use to format the way the APIs are listed:

- Refresh
- Expand All
- Collapse All
- Group by API Registry
- API Presentation
- API Registries. Selecting a registry from this drop-down list toggles between displaying and hiding the registry in the API Explorer.

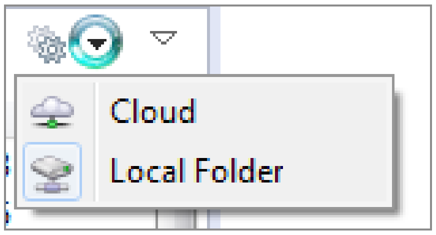
Searching for APIs in API Explorer

Use the search filter that appears at the bottom of the API Explorer view to search for API names that match the string that you enter in the **Filter** text box. You can search by typing in the version number, the full API name, or a full word within an API name. Wildcards is not supported. The search is case insensitive.

Filtering the APIs in the API Explorer view

If your APIs reside in multiple locations and you have set up the API registries in the Settings dialog of the API Modeler view, you can filter the APIs in API Modeler such that it shows you only the APIs available in a certain registry.

To do so, click the (▾) button on the upper right corner of the API Modeler view and select the registry whose APIs you want to view.



Settings

The Settings dialog is configured by default to access the site. You can configure additional registries that you might need to access using this dialog.

API Registry Configurations

To configure a new API registry, click the **New** button. Enter the following information:

name	A unique name for the registry
type	Registry type • **Cloud** - Select the option when the registry is in the cloud and then enter the URL in the **URL** text box. • **Local Folder** - Select the option if the API resides on your local file system, and then browse to the folder using the **Browse** button.
URL	If you selected **Cloud** as your registry type, you must enter the site's URL in this text box.
Authentication	When creating a new API registry, enter your user name and password for the registry that exists on the Cloud.

API Presentation

Configure how you want your API to appear in this view. The three types of presentations available are:

- **Flat** - Displays the APIs as a flat list with each API's version number displayed next to its name in parenthesis. If there are multiple versions of the same API, each version is shown as a separate API, hence multiple APIs with the same name but different version numbers.
- **Hierarchical** - Displays every API as a hierarchy of API name label with version number folder under it and the actual API under the version folder. If there are multiple versions for an API, each version is listed in its own separate folder under the API name label.
- **Latest Version** - If one or more APIs in your registry has multiple versions, selecting this option shows only the latest version of the API and hides the older versions.

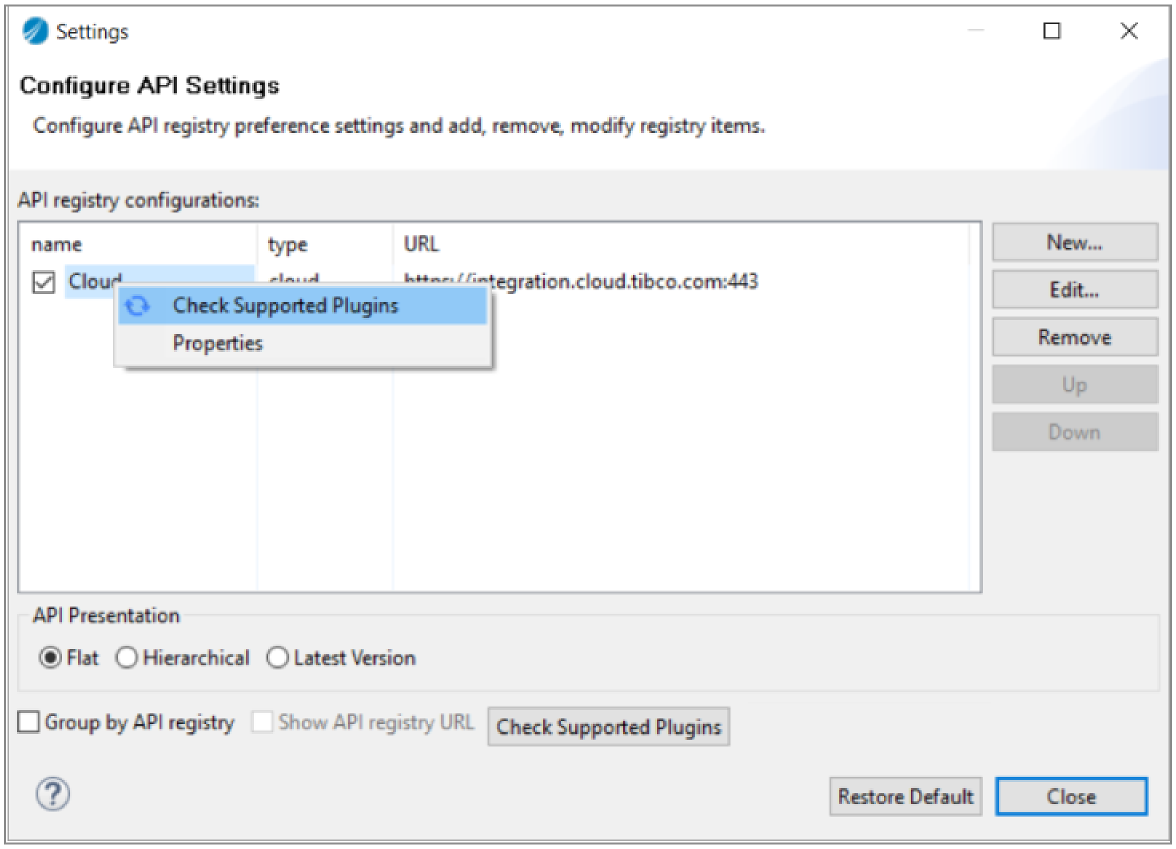
Other Configurations

Group by API registry - Groups the APIs according to the registry from which they were discovered.

Show API Registry URL - Displays the URL of the APIs next to the registry name.

Group apps by sandbox - If you have multiple sandboxes that contain apps, the **Cloud Applications** view displays the sandboxes and groups the apps under their respective sandbox.

Check Supported Plug-ins - This button refreshes the supported list of plug-ins from . When you import an existing project that uses plug-ins, you can validate that the plug-ins used in the project are supported in by clicking this button. A message is displayed indicating that the supported plug-ins are synchronized. The read-only list of supported plug-ins shows up in the **Supported Plug-ins** tab of the **Properties** dialog that you can access from the right-click menu as shown below. You can verify that you have the latest list from the synchronization timestamp at the bottom of the **Properties** dialog. You can also access the **Check Supported Plug-ins** option by right-clicking in the API registry that you want to connect to and selecting it from the resulting context menu.



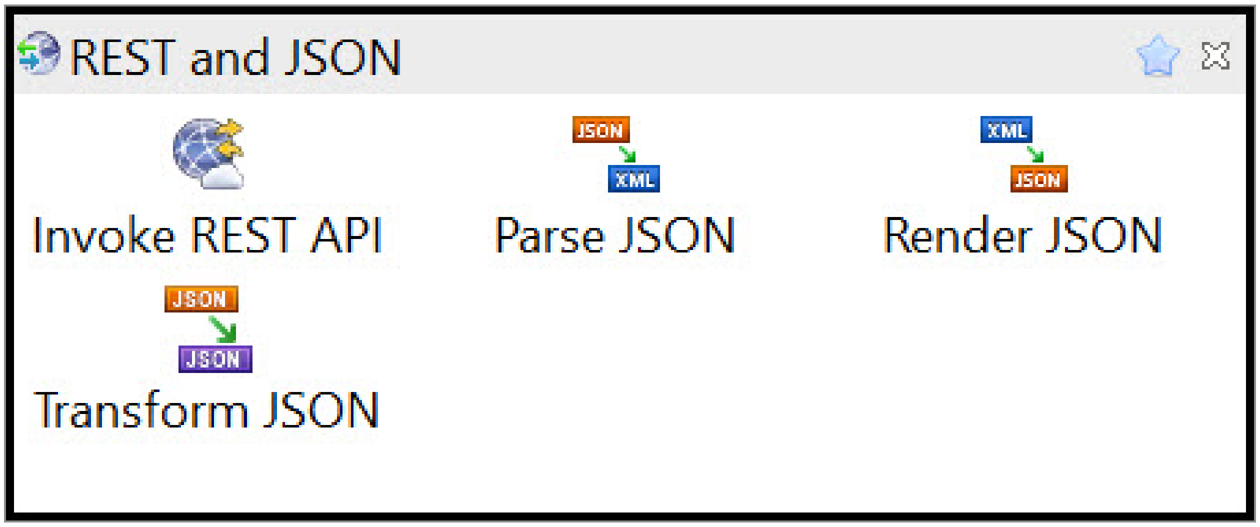
This list represents the plug-ins that is available to your projects in during runtime. In order to use a plug-in during design time, you must have the plug-in installed locally on your machine. If your project uses a plug-in that is not supported in , an error message is displayed while pushing the project to the cloud.

Properties - The **Properties** context menu item opens a dialog which provides information about the registry from which you selected **Properties** in its **General** tab. The **Supported Plug-ins** tab provides a read-only list of plug-ins that are supported in TIBCO BusinessWorks Container Edition.

The REST and JSON Palette in TIBCO Business Studio™ for BusinessWorks™

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

For more information about these activities, see the "REST and JSON" palette topics in *TIBCO BusinessWorks™ Container Edition Bindings and Palettes Reference*.



TIBCO Documentation and Support Services

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

How to Access TIBCO Documentation

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

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

Product-Specific Documentation

The following documentation for this product is available on the TIBCO BusinessWorks™ Container Edition page:

- *TIBCO BusinessWorks™ Container Edition Release Notes*
- *TIBCO BusinessWorks™ Container Edition Installation*
- *TIBCO BusinessWorks™ Container Edition Application Development*
- *TIBCO BusinessWorks™ Container Edition Application Monitoring and Troubleshooting*
- *TIBCO BusinessWorks™ Container Edition Bindings and Palettes Reference*
- *TIBCO BusinessWorks™ Container Edition Concepts*
- *TIBCO BusinessWorks™ Container Edition Error Codes*
- *TIBCO BusinessWorks™ Container Edition Getting Started*
- *TIBCO BusinessWorks™ Container Edition Migration*
- *TIBCO BusinessWorks™ Container Edition Performance Benchmarking and Tuning*
- *TIBCO BusinessWorks™ Container Edition REST Implementation*
- *TIBCO BusinessWorks™ Container Edition Refactoring Best Practices*
- *TIBCO BusinessWorks™ Container Edition Samples*

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

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

How to Join TIBCO Community

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

Legal and Third-Party Notices