Consuming REST APIs by using the new REST request nodes in IBM Integration Bus

Simon Stone
Published on September 27, 2016

In the first release of IBM Integration Bus V10, we introduced support for building new REST APIs projects for exposing integrations via a RESTful interface. This functionality has been extended in subsequent fix packs with “create from scratch,” in the toolkit, mapping support for JSON Schema, and integration with IBM API Connect.

Recently, we have been focused on providing you (the integration developer!) with first class support for consuming REST APIs from message flows. Fix pack 6 introduces a new set of REST request nodes that can be used to consume REST APIs that are defined by a Swagger document:

Introduction

In previous releases of IBM Integration Bus, it was possible to call REST APIs by using the HTTP request nodes. However, for complex REST requests (such as those with path parameters) this can be cumbersome and can require the use of a compute node to manually build URLs and set other Local Environment overrides:

```
SET OutputLocalEnvironment.Destination.HTTP.RequestLine.Method = 'DELETE';
SET OutputLocalEnvironment.Destination.HTTP.RequestURL =
    'http://someserver.ibm.com/api/v1/customers/' ||
    customerId ||
    '/orders/' ||
    orderId ||
    '/items/' ||
    itemId;
SET OutputLocalEnvironment.Destination.HTTP.QueryString.forceDelete = TRUE;
SET OutputLocalEnvironment.Destination.HTTP.QueryString.areYouSure = 'really sure';
SET OutputLocalEnvironment.Destination.HTTP.QueryString.lastChance = 'do it';
```

The REST request nodes make even complex REST requests easy, with node options that allow you to specify parameter values without a preceding compute node. The REST request nodes also offer automatic Content-Type handling, control over request/response body message tree locations, authentication, enhanced activity logging, and an extensive set of Local Environment overrides.

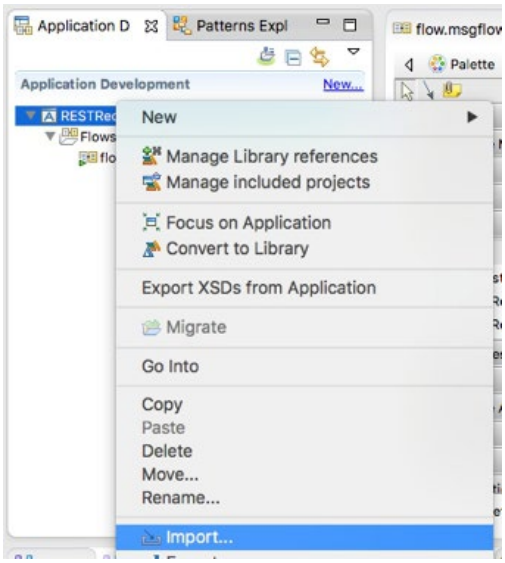
Importing Swagger documents into the REST API Catalog

The REST request node requires a Swagger document that describes the REST API that you want to call. The Swagger document supplies the REST request node with the information that it needs to call operations in the REST API, such as:

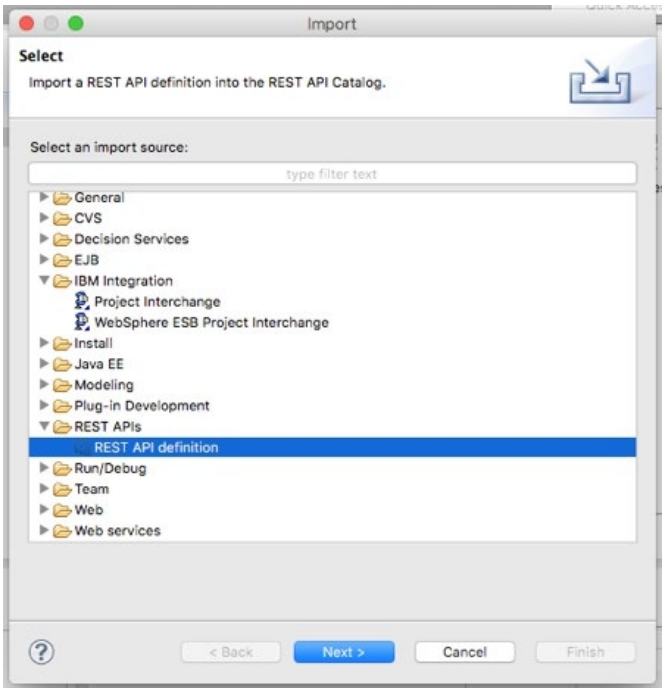
- What URL to use (host name, port, HTTP or HTTPS?)
- What parameters exist for the operation?
- How should those parameters be encoded (path, query, header?)
- What security requirements are in place for the operation?

Swagger documents can come in either JSON or YAML formats. Prior to fix pack 6, IBM Integration Bus only supported Swagger documents in the JSON format, but fix pack 6 introduces additional support for the YAML format.

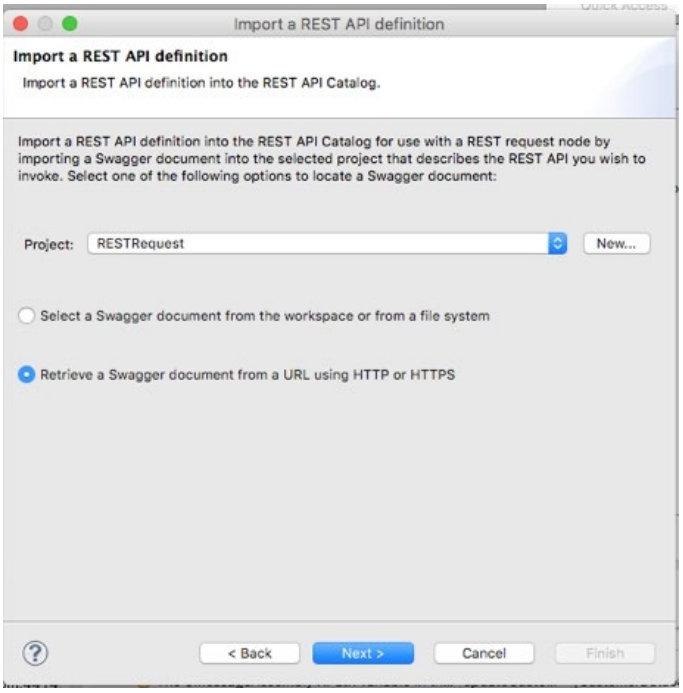
Swagger documents can be imported into your projects (such as applications or shared libraries) by using a new import wizard. The wizard is available by right clicking the project and clicking “Import...”, or by clicking on “File”, “Import...” in the menu:



Next, click on “REST APIs”, and then “REST API definition”:

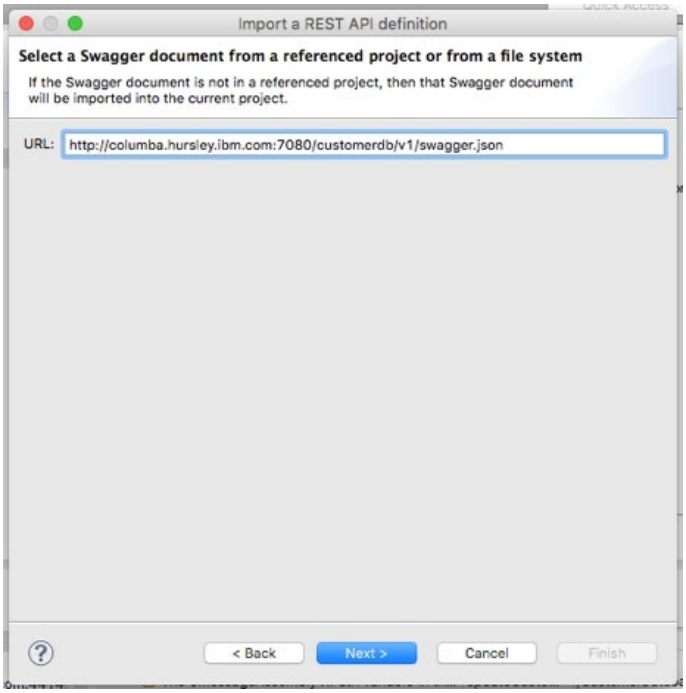


The “Import a REST API definition” wizard appears. You can choose to select a Swagger document from the file system, such as one you have downloaded, or you can retrieve a Swagger document from a URL:

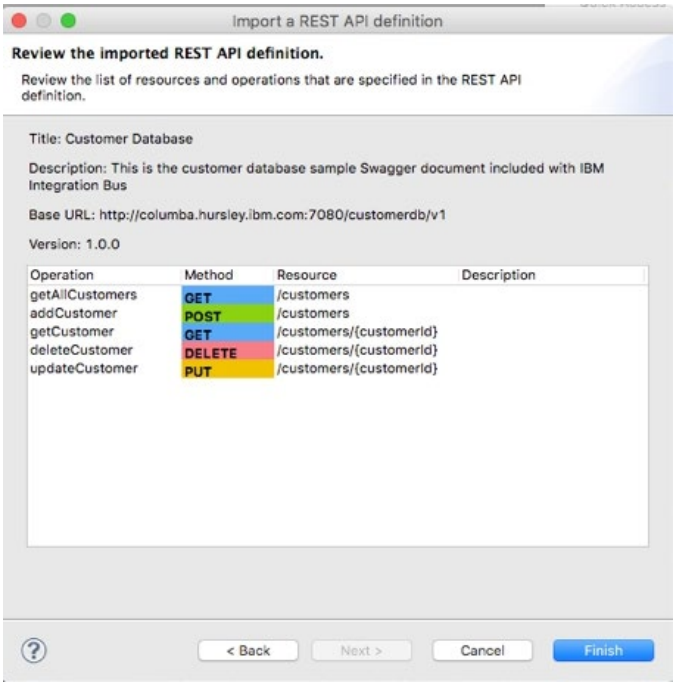


I am going to use a Swagger document from a REST API that I have deployed to another integration node; a link to the Swagger document is available from the web user interface for that integration node:

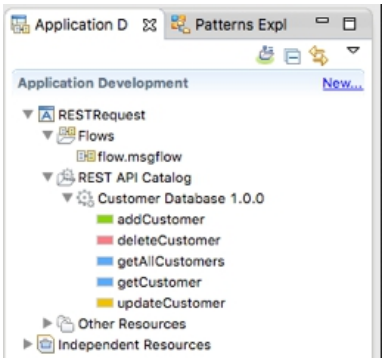
I can paste that URL into the wizard and click “Next”:



When I click “Next”, the toolkit downloads the Swagger document from the specified URL, and shows me the operations that are available:



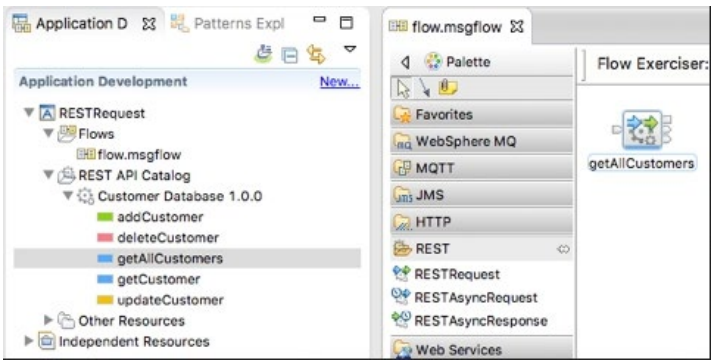
Once I click “Finish”, the Swagger document is saved into the project. Swagger documents in projects are automatically loaded into the “REST API Catalog”. The “REST API Catalog” is a new category that appears in the project and shows all REST APIs and operations that are available to invoke in the project:



Alternatively, you can import Swagger documents into a project by dropping a new RESTRequest or RESTAsyncRequest node on the message flows canvas. When a new REST request node is dropped onto the canvas, the “Invoke an operation in a REST API” wizard appears, which allows you to select or retrieve a Swagger document for that REST request node.

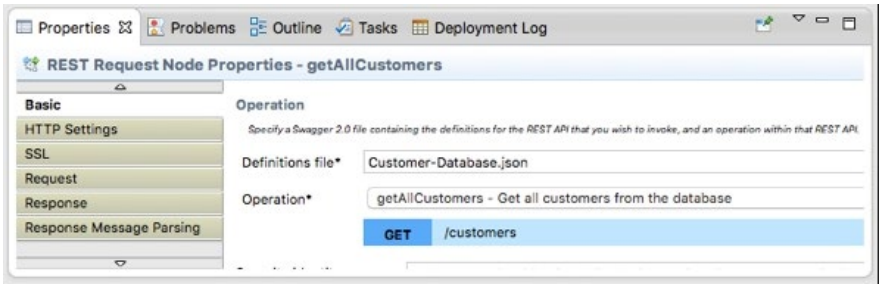
Calling an operation available in the REST API Catalog

Once a REST API is shown in the REST API Catalog, the operations in that REST API can be dragged from the REST API Catalog onto the message flow canvas. This action creates a new RESTRequest node that is automatically configured to call the dragged operation:



The new RESTRequest node is automatically named after the selected operation. The Swagger document and operation name are

configured on the node properties:



You can reconfigure an existing RESTRequest or RESTAsyncRequest node by dragging and dropping an operation from the REST API Catalog onto that node. You can also use the node properties to change the selected Swagger document and operation. When you modify the selected operation, the node name is automatically renamed, but not if the node name has been manually specified.

For REST API operations that do not take any parameters, this is the only configuration that is required. The REST request nodes will use the information in the Swagger document to call the selected operation, using the input message as the request body (if the HTTP method permits a request body) and placing the response body into the output message as per other message flow nodes.

Supplying parameter values for an operation

Many operations in REST APIs accept one or more parameters. There are three types of parameters that are supported by IBM Integration Bus:

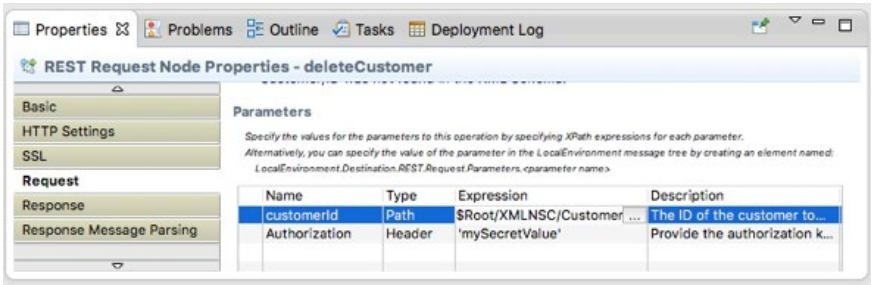
Path parameters – where part of the path used to form the URL is variable, such as the “customerId” parameter in the path “/customers/{customerId}”.

Query parameters – key/value pairs that are appended to the URL, for example the “max” parameter in the URL “http://localhost:7800/customerdb/v1/customers?max=5”.

Header parameters – HTTP headers that are sent in the HTTP request to the REST API, for example the “X-IBM-Client-ID” parameter “X-IBM-Client-ID: 4f059bfc-db36-49f7-908a-42cfa9787af6”.

The REST request nodes can be passed values for these parameters, and the REST request node will handle encoding the parameter values into the HTTP request.

The “Parameters” table is available on the “Request” tab of the node properties. This table shows all of the parameters that are available for the selected operation:



The table shows the name, type, and a description of the parameter. The “Expression” column is editable, and can be used to pass in parameter values. Supported values include XPath string, number, or boolean literals (for example, a string literal is used for the “Authorization” parameter) or XPath/ESQL expressions that refer to data in the input message (for example, an XPath expression into an XMLNSC message is used for the “customerId” parameter).

At runtime, the expressions are evaluated to determine the value that should be used for each parameter when making the request to the REST API operation.

Alternatively, parameter values can be specified in the Local Environment. A new LocalEnvironment.Destination.REST.Request.Parameters.<parameter name> section has been added. This allows parameter values to be specified programmatically in a compute node, or graphically in a mapping node.

Any parameter values specified in the Local Environment override parameter values specified as node properties in the parameters table. If parameter values are specified in the Local Environment, then the “Expression” value on the parameters table can be left blank. However, if the Swagger document specifies that the parameter is required, but a value is not specified in the Local Environment or on the node properties, then a BIP6360E error message will be thrown.

MIME types and automatic Content-Type handling

The REST request nodes include full support for Content-Type and Accept header handling. These headers contain MIME types that determine the format/type of the data stored in the request or response body. Users that call REST APIs which support multiple data formats (such as JSON and XML) may need to specify the value of these headers to select which data format they want.

Content-Type

The “Content-Type” property on the “Request” tab determines the MIME type that is set as the value of the “Content-Type” header in the HTTP. This header specifies the format of the data in the HTTP request body. By default, the REST request node selects a MIME type based on the input message – “application/json” for JSON messages, and “application/xml” for XMLNSC messages:

Alternatively, you can manually specify a MIME type:

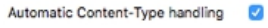
Accept

The “Accept” property on the “Response” tab determines the MIME type that is set as the value of the “Accept” header in the HTTP request. This header is a “hint” to the REST API as to what format of data is expected in the HTTP response body from that REST API. Please note that REST APIs may not respect the value of the “Accept” header! By default, the REST request node specifies the special MIME type “*/*”, which means “give me the default format”:

Alternatively, you can manually specify a MIME type:

Automatic Content-Type handling

The “Automatic Content-Type handling” property on the “Response Message Parsing” tab enables functionality in the REST request node that automatically selects the message domain to use:



When this functionality is enabled, the REST request node inspects the “Content-Type” header in the HTTP response from the REST API and matches it against a suitable message domain. For example, if the REST API returns a “Content-Type” header of “application/json”, then the REST request node will select the JSON message domain.

This property is enabled by default. When the property is enabled, you cannot manually specify a message domain on the node properties. You must disable the property first.

Control over request and response body locations

By default, the REST request node uses the whole of the input message (InputRoot) as the HTTP request body. Additionally, the HTTP response body replaces the contents of the input message and is set as the contents of the output message (OutputRoot).

The REST request node can be configured to extract the HTTP request body from any location in the input message, including the Local Environment or Global Environment message trees. This configuration can be performed by specifying an XPath expression in the “Input body location”.

For example, if I want to use an XMLNSC message I have built in the Local Environment as the HTTP request body, I would use an XPath

expression such as:

The REST request node can also be configured to place the HTTP response body in any location in the output message, whilst preserving the input message. This functionality can be useful when calling a set of micro services, where the HTTP responses can be collected in the Local Environment tree for aggregation by a later compute or graphical mapping node. This configuration can be performed by specifying an XPath expression in the “Output body location” property.

For example, if I want to store the HTTP response body into the Local Environment, I would use an XPath expression such as:

The “Error body location” property is used when the REST API returns a non-successful HTTP status code (4xx or 5xx), and a message is propagated out of the “Error” terminal:

Finally, it is possible to place only a portion of the HTTP response body into the location specified by the “Output body location” property. This functionality can be useful when a micro service returns a large response message, but you only require one small portion of that message, or a single field. This configuration can be performed by specifying an XPath expression in the “Response body location”.

For example, if I want to extract just the “name” field from a JSON response message, I would use an XPath expression such as:

Authentication

A lot of REST APIs that IBM Integration Bus users will want to call are likely to be secured in a way that requires the REST request nodes to authenticate to the remote HTTP server. For example, a REST API might be secured with:

- HTTP Basic Authentication (username + password)
- API authentication key passed as a header or query parameter
- SSL/TLS client or mutual authentication

Configuration for SSL/TLS client and mutual authentication can be found in the “SSL” tab of the node properties, and all of the standard SSL/TLS properties from other message flow nodes are available.

The HTTP request nodes support HTTP Basic Authentication, but it is not a simple experience. The options include placing usernames and passwords into the Properties message tree, or using security policies, or building your own “Authorization” header.

The REST request node improves this by providing support for automatically handling HTTP Basic Authentication and API authentication keys using security credentials defined using the familiar “mqsisetdbparms” command.

This support requires that the Swagger document includes information on the security requirements for the REST APIs. Refer to the definitions of the “Security Definitions Object”, “Security Requirement Object”, and “Security Scheme Object” in the Swagger 2.0 specification for further details. “Security Scheme Objects” with a type of “basic” or “apiKey” are supported by IBM Integration Bus. “Security Scheme Objects” with a type of “oauth2” are not currently supported by IBM Integration Bus.

A username and password can be associated with a security identity:

```
mqsisetdbparms TESTNODE_sstone1 -n rest::myBasicAuth -u myUser -p myPassword
```

An API key can also be associated with a security identity:

```
mqsisetdbparms TESTNODE_sstone1 -n rest::myApiKey -k 32B8976C-1C90-494F-AC7B-17AE418EDF51
```

A combination of the two is also possible:

```
mqsisetdbparms TESTNODE_sstone1 -n rest::myBasicAuthAndApiKey -u myUser -p myPassword -k 32B8976C-1C90-494F-AC7B-17AE418EDF51
```

The security identity can then be specified on the REST request node, as the value of the “Security identity” property on the “Basic” tab. The REST request node will then use the stored security credentials and add them into the HTTP request as specified in the Swagger document:

Activity logging

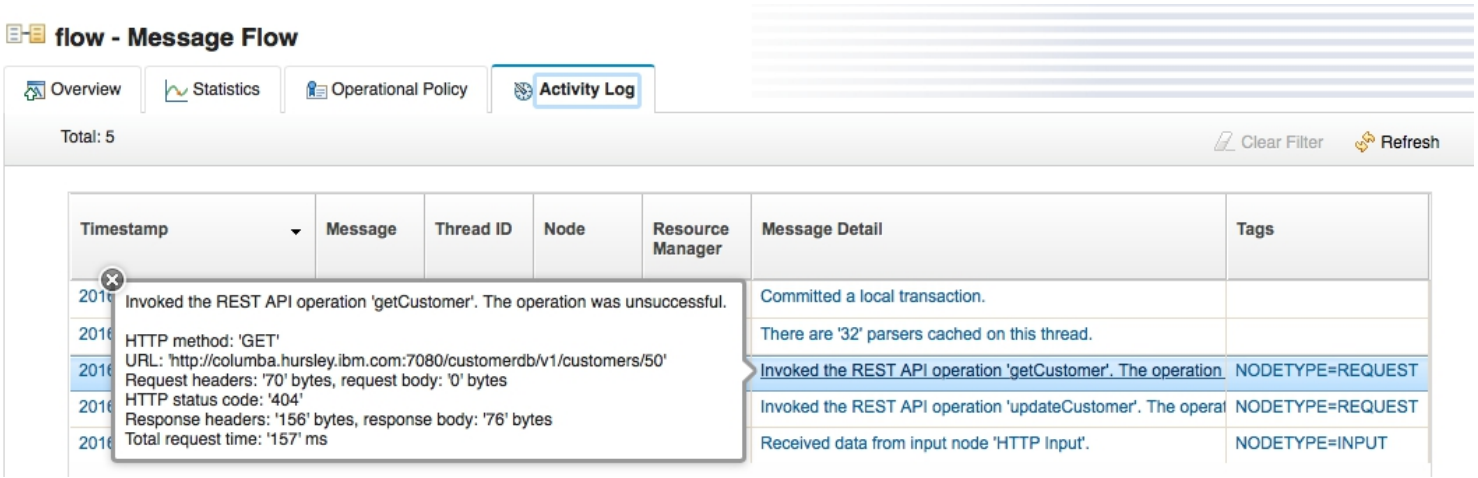
The REST request nodes write information on all REST requests, whether those REST requests are successful or unsuccessful, to activity log. Activity log is always enabled, and can be used to record interactions with REST APIs, and diagnose problems with REST requests. Activity log entries are kept in integration server memory by default, and are accessible via the web administration user interface, REST APIs, and Integration Java API. They can also be written to file by defining an “ActivityLog” configurable service.

The REST request nodes write activity log entries that include:

- The operation name, HTTP method, and URL used to call the REST API.
- The size of the HTTP request headers and body sent to the REST API.
- The size of the HTTP response headers and body received from the REST API.
- The HTTP status code from the REST API.
- The total time spent waiting for the response from the REST API.

The total request time differs from the information available for the REST request nodes via accounting and statistics, because the total request time only includes the time spent calling the operating system send and receive functions. The total request time does not include time spent elsewhere in the REST request node, for example parsing of the HTTP response message. Unreasonably high total request times are indicative of network or REST API performance problems rather than a problem with the message flow containing the REST request node.

Activity log can be viewed in the web administration user interface by navigating to the message flow that contains the REST request node, and clicking on the “Activity Log” tab:



Local Environment overrides and output properties

The REST request nodes support an extensive set of Local Environment overrides that can be used to dynamically change the configuration of the REST request node at runtime. These Local Environment overrides include:

- The operation name.
- The values for parameters to the operation.
- The values of the “Content-Type” and “Accept” node properties.
- The value of the “Security identity” node property.
- Usernames, passwords, and API keys can also be specified.

The value of the “Base URL” node property.

Additionally, the REST request nodes write a set of properties to the Local Environment output by the node. These output properties include:

The operation name, HTTP method, and URL used to call the REST API.

The size of the HTTP request headers and body sent to the REST API.

The size of the HTTP response headers and body received from the REST API.

The HTTP status code from the REST API.

The total time spent waiting for the response from the REST API.

The full set of Local Environment overrides and output properties are documented in the Knowledge Center:

Conclusion

In this blog post, you have seen how you can use the new REST request nodes introduced in 10.0.0.6 to invoke REST APIs, where those REST APIs are described with a Swagger document. Look out for future blog posts on the REST request node, including a post on how you can use the graphical mapper in conjunction with the REST request nodes.

TAGS REST APIS

Simon Stone

4 comments on"Consuming REST APIs by using the new REST request nodes in IBM Integration Bus"

Soraya January 01, 2020

When I use the rest request node I got this error
The rest request node requires the attribute “definitionType”
I can´t find this attribute anywhere.

Can you please help me? Thanks

Reply (Edit)

Jogwar May 25, 2017

I understand there are three steps to make the Basic Security work.
1. Create a swagger file with the above security definitions (and label)
2. Using mqsisetdbparams, create the security identity
3. use the label in the REST Request node, configurable parameter ‘Security Identity’

Unfortunately, even after doing these three things, I am getting 401 error. Here is what I am using in a swagger file.
“basePath” : “/services/ws/UserSearch”,
“schemes” : [“https”],

“securityDefinitions” : {
“UserProfileCreds” : {
“type”: “basic”
}
},

“paths” : {
“/{type}/{id}” : {
“get” : {
“operationId” : “getProfile”,
“security” : [
{ “UserProfileCreds”: [ ] }
],
“responses” : {
“200” : {
“description” : “The operation was successful.”,
“schema” : {
“type” : “string”
}

Reply (Edit)

Gerald Kallas January 23, 2017

I you want to use basic authentication you need to add 2 sections to the Swagger file ..

1st ..

“securityDefinitions” : {
“CHECKMOBILE” : {
“type”: “basic”
}
}

at top level, 2nd ..

“security” : [
{ “CHECKMOBILE”: [ ] }
]

at path level. In this case the “CHECKMOBILE” label defined w/ mqsisetdbparams will be leveraged.

Reply (Edit)

Narendra October 02, 2016

Hell Simon.
How to handle Form encoded Type parameters in post method.
below four fields are form encoded in my restrequest node.
datacontent
contenttype
contentformat

contentencoding

Reply (Edit)