Using some of the more advanced features of the LoopBackRequest node

GREGH
Published on September 13, 2016 / Updated on September 19, 2016

0

Introduction

IBM Integration Bus v10.0.0.6 provides the LoopBackRequest node which is a new node that allows flow developers to easily create, retrieve, update and delete data records in external systems such as MongoDB, Cloudant and PostgreSQL.

In my previous post, a Basic Introduction to Using the LoopBackRequest node, I provided a simple example that retrieved data from a MongoDB database which explained how to:

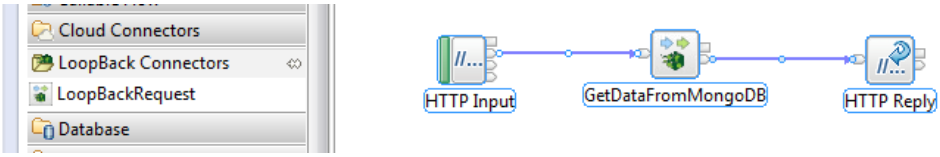
- Install LoopBack connectors.
- Configure connections to backend systems using LoopBack datasources.
- Use the LoopBackRequest node to retrieve data from a backend system.

In this post I will illustrate some of the more advanced features that are available with the LoopBackRequest node. To facilitate that, we will extend the message flow that was used in the first article to:

- Use LocalEnvironment filters to select a subset of the documents from a MongoDB SHAREPRICE collection.
- Install and configure a second LoopBack connector to enable connections to a Cloudant database running as a service in IBM Bluemix.
- Use some of the information from the matching SHAREPRICE documents retrieved from MongoDB to create COMPANY documents in a Cloudant database called ‘demodb‘, using another LoopBackRequest node.
- Use a LoopBack model to retrieve the COMPANY documents in a specified order from the Cloudant database.

The Initial Message Flow

We’ll use the message flow that was developed in the first article as a starting point for this example. Just to recap, this message flow was used to retrieve all of the documents from the SHAREPRICE collection in a MongoDB database called ‘loopback’ and it looked like this:



Use of the LoopBackRequest node in a message flow

This message flow was used to retrieve the following SHAREPRICE documents.

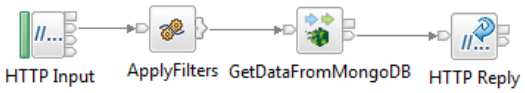
```
{ "_id" : ObjectId("576952b71181b9fc27121165"), "companyId" : "100", "company" : "Barclays PLC", "phone"
: "0800 400100", "price": 171 }
{ "_id" : ObjectId("576952b71181b9fc27121166"), "companyId" : "200", "company" : "HSBC Holdings PLC",
"phone" : "0345 740 4404", "price" : 430 }

{ "_id" : ObjectId("576952b71181b9fc27121167"), "companyId" : "300", "company" : "Prudential PLC",
"phone" : "0800 000 000", "price" : 1298 }

{ "_id" : ObjectId("576952b71181b9fc27121168"), "companyId" : "400", "company" : "Worldpay Group PLC",
```

```
"phone" : "0808 253 0870", "price" : 261 }

{ "_id" : ObjectId("576952b71181b9fc27121169"), "companyId" : "500", "company" : "Aviva Insurance PLC",
"phone" : "0800 142142", "price" : 440 }

{ "_id" : ObjectId("576952b71181b9fc2712116a"), "companyId" : "600", "company" : "Eastspring
Investments", "phone" : "0800 000 000", "price" : 325 }
```

Using LoopBack LocalEnvironment Values to Filter The Returned Documents

In this example, we are going to specify a 'where' filter to select only the SHAREPRICE documents with a 'price' greater than or equal to 325 and a 'phone' number that begins with '08'. Those documents are highlighted above in green. Additionally, for the purposes of illustrating a second filtering option, we are only interested in retrieving the 'company' and 'phone' fields from these documents so we will also add a 'field' filter to select only those fields.

LoopBack filters are set in the LocalEnvironment.Destination.Loopback.Request folder and one way to set these is to use a Mapping Node as explained in the Interacting with MongoDB using IBM Integration Bus LoopBackRequest node article. In this example, we will use ESQL in a Compute node, which I've called 'ApplyFilters', to provide an alternative method.



Amended flow with ApplyFilter Compute node added.

Because the ESQL updates values in the LocalEnvironment we must ensure that the 'Compute mode' property on the 'Basic' tab of the 'ApplyFilters' node is set to 'LocalEnvironment and Message'. This ensures that these values are visible to and can be used by the 'GetDataFromMongoDB' LoopBackRequest node.

Setting a WHERE clause filter

To select only a subset of the documents from the SHAREPRICE collection we can set a LoopBack Node API 'where' clause in OutputLocalEnvironment.Destination.Loopback.Request.filter.where. You can find more information on using the LoopBack Node API 'where clause for queries' here.

In this case we are combining two selection conditions with the 'and' operator which has the following syntax:

`{where: {<and|or>: [condition1, condition2, …] } }`

`'condition1'` is to select documents with a 'price' greater than or equal to 325 so we'll use the 'gte' operator.

```
{"price": {"gte":325}}
```

`'condition2'` is to select documents with a 'phone' number that begins with '08' and for this we'll use the regular expression operation 'regexp'.

```
{"phone": {"regexp": "^08"}}
```

When setting a where clause in LocalEnvironment.Destination.Looback.Request.filter.where the outermost enclosing `{where: ….. }` part of the clause is added for you so this leaves us with the following ESQL to set the required where clause:

```
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.where = '{"and": [{"price": {"gte":325}},
{"phone": {"regexp": "^08"}}]}';
```

Setting a ‘fields’ filter

A LoopBack Node API ‘field’ filter can be used to select specific fields from a document. More information on this can be found in the LoopBack documentation here. A field filter has the following syntax:

```
{ fields: {propertyName: <true|false>, propertyName: <true|false>, … } }
```

In this example, we want to retrieve only the ‘company’ and ‘phone’ fields from our SHAREPRICE documents and to achieve this we will set the field filter as a list in the LocalEnvironment, as follows:

```
SET OutputLocalEnvironment.Destination.Loopback.Request
.filter.field[1].company = true;
SET OutputLocalEnvironment.Destination.Loopback.Request
.filter.field[2].phone = true;
```

The lines above set a fields filter to explicitly include the ‘company’ and ‘phone’ fields. When an ‘explicit’ filter is set in this way the MongoDB connector excludes selecting all other fields in the document with the exception of the ‘id’ field. For this reason, I have specified a third field filter to explicitly exclude the ‘id’ field.

```
SET OutputLocalEnvironment.Destination.Loopback.Request
.filter.field[3].id = false;
```

The ESQL to achieve the filtering that we want in our ‘ApplyFilters’ Compute node therefore ends up as below:

```
-- Apply a 'where filter' to only retrieve SHAREPRICE documents with a 'price' value >= 325 and a
'phone' number that begins with 08;
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.where = '{"and": [{"price":
{"gte":325}}, {"phone": {"regexp": "^08"}}]}';

-- Apply a 'fields filter' to only get the 'company' and 'phone' fields from the matching
SHAREPRICE documents
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.field[1].company = true;
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.field[2].phone = true;

-- Note : the loopback-connector-mongodb connector always returns the 'id' field so we have to
-- explicitly exclude it if we don't want it
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.field[3].id = false;
```

The ESQL in the ‘ApplyFilters‘ Compute node

To confirm the filters are working as expected we’ll once again use the ‘Flow Exerciser’ to test the message flow and processing a message through the flow produces the following JSON array in the output message:

```
[
     {“company”:”Prudential PLC”,”phone”:”0800 000 000”},
     {“company”:”Aviva Insurance PLC”,”phone”:”0800 142142”},
     {“company”:”Eastspring Investments”,”phone”:”0800 000 000”}
]
```

Preparing the Integration Node runtime environment to connect to Cloudant

Having retrieved a subset of SHAREPRICE documents from the MongoDB database our next requirement is to create a copy of those documents in a Cloudant NoSQL database, called demodb, which is running as a service in IBM Bluemix. To do this we will need to setup the Integration Node’s runtime environment with a suitable LoopBack ‘Cloudant’ connector and configure the datasources.json file with connection details for the Cloudant database.

In my earlier article I showed how to install LoopBack connectors into the Integration Node runtime environment using the `‘npm’` command. To connect to the Cloudant database I will use the LoopBack Cloudant connector which must be installed into the MQSI_WORKPATH/node_modules directory.

```
cd %MQSI_WORKPATH%\node_modules
npm install loopback-connector-cloudant -save
```

The documentation for the LoopBack Cloudant connector explains how to configure a connection to a Cloudant datasource and lists the following connection parameters:

database (String) – Database name

username (String) – Cloudant username, use either 'url' or username/password

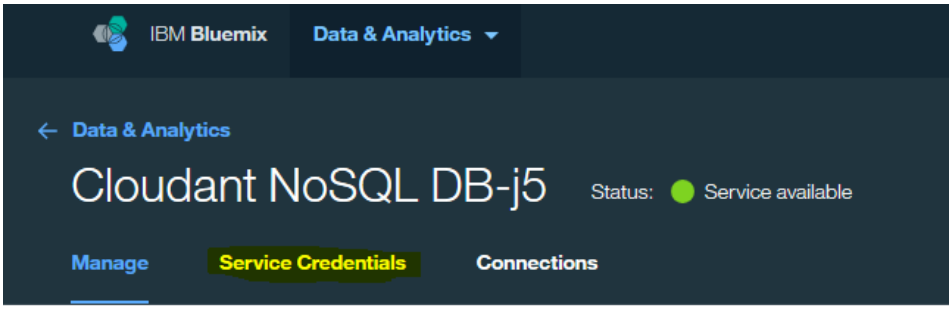
password (String) – Cloudant password

url (String) – Cloudant URL containing both username and password

modelIndex (String) – Specify the model name to document mapping, defaults to 'loopback__model__name'

In this example, I will use `'mqsisetdbparms'` to create a 'Security Identity' called CLOUDANT_SEC_ID which will configure the username and password for the Cloudant connection. I've taken this approach to avoid having these credentials shown in plain text in the datasources.json file.

To determine the username and password I logged into IBM Bluemix, selected the 'Cloudant service' and then selected the 'Service Credentials' menu option:



Cloudant Service Credentials Menu

A panel is then displayed showing the connection credentials (which I have obscured):

Cloudant Service Credentials Panel

With this information to hand we can setup the CLOUDANT_SEC_ID Security Identity with `mqsisetdbparms`, replacing <username> and <password> with the values from the panel above:

```
mqsisetdbparms TESTNODE_hattg -n loopback::CLOUDANT_SEC_ID -u <username> -p <password>
```

With the username and password set in this way the configuration I need to add for the CLOUDANT datasource in the /connectors/loopback/

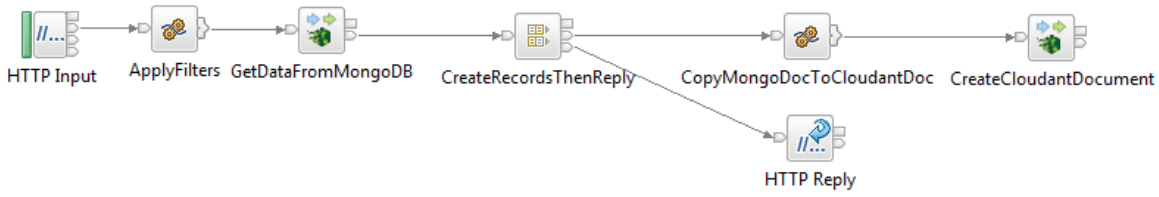
datasources.json file is:

```
{
  "MONGO": {
     "host": "localhost",
     "port": 27017,
     "database": "loopback",
     "name": "MONGO",
     "connector": "mongodb"
  },
  "CLOUDANT": {
     "database": "demodb",
     "name": "CLOUDANT",
     "port": 443,
     "connector": "cloudant"
  }
}
```

Updated datasources.json with CLOUDANT datasource

Extending the message flow to create documents in Cloudant

To create the documents in the Cloudant database we will add three nodes to our message flow so that it looks like this:



Extended message flow to create Cloudant documents

The 'CreateRecordsThenReply' node is a FlowOrder node which I have introduced so that I can process all the documents that were returned from MongoDB and create them in Cloudant before sending an HTTP reply.
The 'CopyMongoDocToCloudantDoc' ESQL Compute node is used to iterate over the documents that were returned from MongoDB. Each of the MongoDB documents are copied into a JSON object which is then passed to the 'CreateCloudantDocument' LoopBackRequest node which issues the request to Cloudant to create the COMPANY document.

The ESQL that I used to do this is shown below:

```
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
	CALL CopyMessageHeaders();

	DECLARE cursor REFERENCE TO InputRoot.JSON.Data;

	-- If we have been returned a JSON array move to the first 'Item' in it
	IF (FIELDTYPE(cursor.Item[1]) IS NOT NULL) THEN
		MOVE cursor FIRSTCHILD TYPE Name NAME 'Item';
	END IF;

	WHILE LASTMOVE(cursor) DO
		-- Only create documents when we have been give some 'company' data
		IF (FIELDTYPE(cursor.company) IS NOT NULL) THEN
			CREATE LASTCHILD OF OutputLocalEnvironment.Variables DOMAIN('JSON') NAME 'JSON';
			SET OutputLocalEnvironment.Variables.JSON.Data.companyName = cursor.company;
			SET OutputLocalEnvironment.Variables.JSON.Data.phoneNumber = cursor.phone;
			PROPAGATE;
		END IF;

		MOVE cursor NEXTSIBLING;
	END WHILE;

	RETURN FALSE;

END;
```

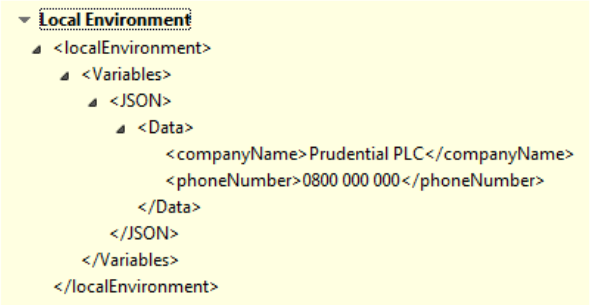
The SHAREPRICE documents retrieved from MongoDB by the LoopBackRequest node are placed into the message tree as a JSON array under the Root.JSON.Data folder. The LoopBackRequest node can return documents as a JSON object for certain types of query (for example, a query by 'id') so I have included a check to cater for that:

```
IF (FIELDTYPE(cursor.Item[1]) IS NOT NULL) THEN
     MOVE cursor FIRSTCHILD TYPE Name NAME 'Item';
END IF;
```

The WHILE loop then iterates over each of the returned documents and copies values from them into a folder in the OutputLocalEnvironment before propagating to the 'CreateCloudantDocument' LoopBackRequest node:

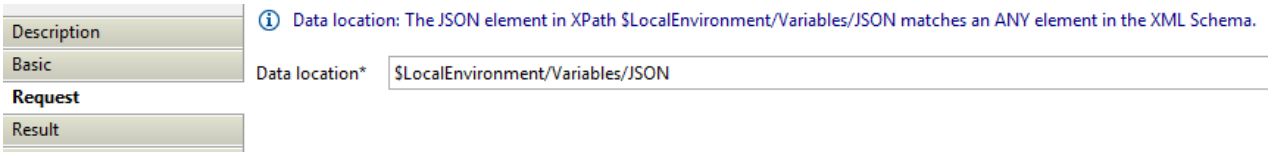
```
CREATE LASTCHILD OF OutputLocalEnvironment.Variables DOMAIN('JSON') NAME 'JSON';
SET OutputLocalEnvironment.Variables.JSON.Data.companyName = cursor.company;
SET OutputLocalEnvironment.Variables.JSON.Data.phoneNumber = cursor.phone;
PROPAGATE;
```

In this example, I have created fields called 'companyName' and 'phoneNumber' whose values are taken from the 'company' and 'phone' fields in the SHAREPRICE documents respectively. This will result in the LocalEnvironment containing a JSON object in the Variable subfolder structured like this:



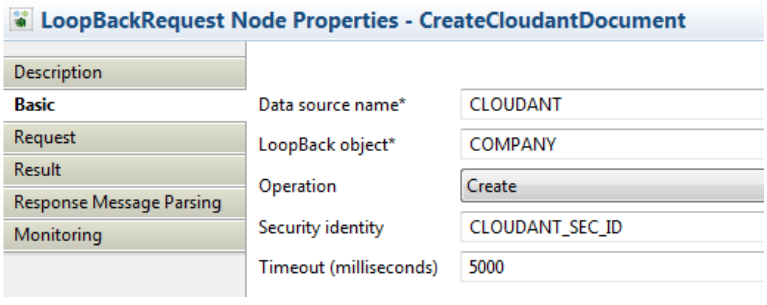
Example of the COMPANY document in the LocalEnvironment

There's no real necessity to do this other than I wanted to illustrate the 'Data location' property that the LoopBackRequest node provides on its 'Request' tab. We will set the value of this property on the 'CreateCloudantDocument' LoopBackRequest node to $LocalEnvironment/Variables/JSON so that it sends this data in its request to create the COMPANY document in Cloudant.



The 'Data location' property of the LoopBackRequest node.

The properties on the 'Basic' tab of this node are set to use the CLOUDANT datasource that was configured earlier and take the connection credentials from the CLOUDANT_SEC_ID Security Identity that was setup with `'mqsisetdbparms'`.



LoopBackRequest node properties to create a document in Cloudant

The ‘LoopBack object’ property is set to ‘COMPANY’ as those are the document types that we want to create in the Cloudant database which is achieved by setting the ‘Operation’ to ‘Create’.

After wiring the ‘second’ terminal of the ‘CreateRecordsAndThenReply’ FlowOrder node to the HTTPReply node we can once again use the Flow Exerciser to test the updated message flow. Doing this should result in COMPANY documents, similar to the following, being created in the Cloudant database.


```
{
  "_id": "772bf078e783089e1fcc230cf5886b25",
  "_rev": "1-f598125a87fff9c349cacd6a47401feb",
  "companyName": "Prudential PLC",
  "phoneNumber": "0800 000 000",
  "loopback__model__name": "COMPANY"
}
{
  "_id": "94d45569d618208c01b521d50081812a",
  "_rev": "1-d6e7eed9bb81a53eb0c12a7f3c87028d",
  "companyName": "Eastspring Investments",
  "phoneNumber": "0800 000 000",
  "loopback__model__name": "COMPANY"
}
{
  "_id": "a47b2dd7634314488f37a3f22db22419",
  "_rev": "1-d56ab44da337ca79a1e75419edf8ca22",
  "companyName": "Aviva Insurance PLC",
  "phoneNumber": "0800 142142",
  "loopback__model__name": "COMPANY"
}
```

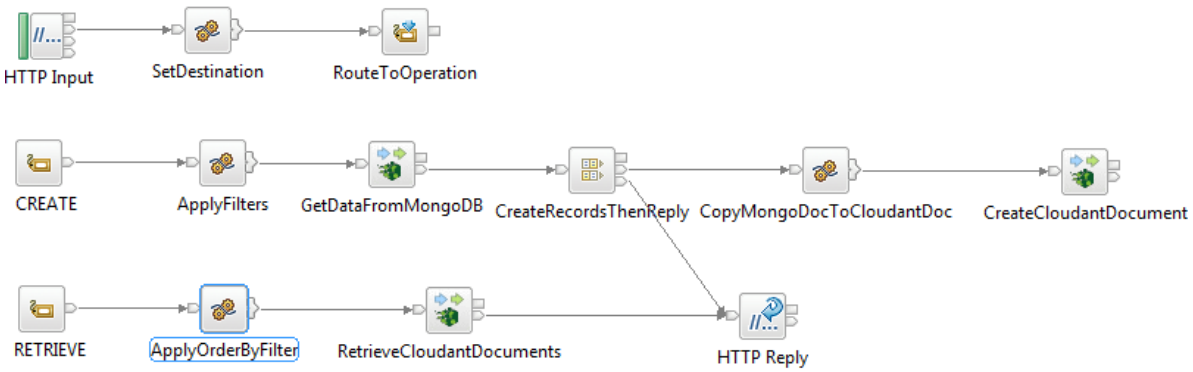

Cloudant COMPANY documents

These documents contain a field field called ‘loopback__model__name’ with a value of ‘COMPANY’ which is the value that the ‘LoopBack object’ node property was set to. This field is added by the LoopBack Cloudant connector and is effectively the document type. The name of this field can be changed using the ‘modelIndex’ property which is explained further in the documentation for the Cloudant connector.

Retrieving Ordered Documents from Cloudant

The last aim for this example was to retrieve the COMPANY documents that were created in Cloudant in a specifc order. The approach to achieve this is to use the LocalEnvironment to set an ‘order’ clause in the LocalEnvironment.Destination.Loopback.request.order folder and once again we’ll use an ESQL Compute node to do this.

At this stage, I decided to do a bit of restructuring of the message flow so that it now looks like this:



Message flow to create and retrieve COMPANY documents from Cloudant

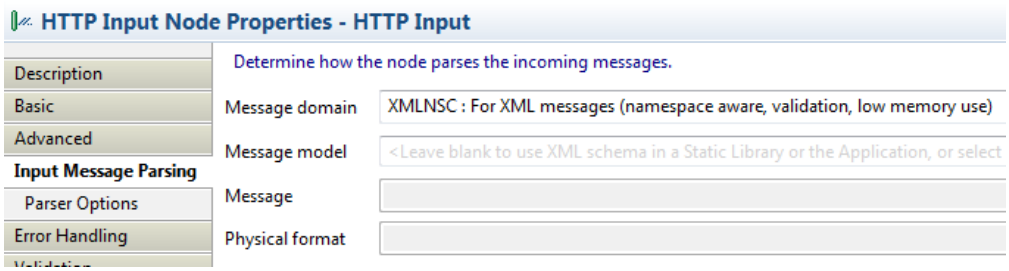
It looks a little different from how it was after we made the changes to create the documents in CLOUDANT but essentially the structure has changed to use a RouteToLabel node called 'RouteToOperation' which uses a field in the input message to decide whether to perform a 'create' or a 'retrieve' by routing the message to either the 'CREATE' of 'RETRIEVE' label nodes.

The following ESQL to set up this routing is used in the 'SetDestination' ESQL Compute node:

```
SET OutputLocalEnvironment.Destination.RouterList.DestinationData
."label" = InputBody.Message.RequestedOperation;
```

The 'Compute mode' property of this node must be set to 'LocalEnvironment and message' because the routing values are set in the LocalEnvironment and these need to be passed downstream for use in the 'RouteToOperation' node.

I've also updated the HTTP Input node so that it parses the input message as XML to facilitate choosing whether to create or retrieve documents based on values in the input message (InputBody.Message.RequestedOperation):



HTTPInput node parser properties

I can now choose to create documents in Cloudant by sending the following input message to the flow:

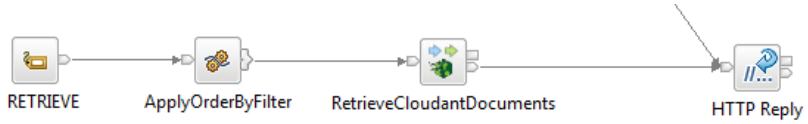
```
<Message><RequestedOperation>CREATE</RequestedOperation></Message>
```

To route the message to the RETRIEVE Label node we will use this input message:

```
<Message><RequestedOperation>RETRIEVE</RequestedOperation></Message>
```

Setting Ordering Criteria

The part of the message flow that performs the retrieval of documents from the Cloudant database is shown below:



Cloudant retrieve part of message flow

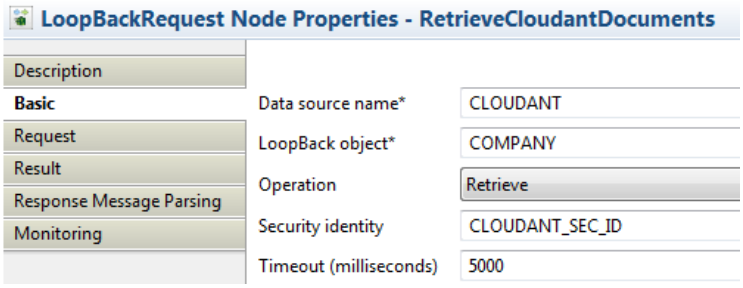
I have used the 'ApplyOrderByFilter' ESQL Compute node, with its 'Compute mode' property set to 'LocalEnvironment and message', to specify an 'order' clause in the LocalEnvironment. This order by clause will then be used by the 'RetrieveCloudantDocuments' LoopBackRequest node. The ESQL I used to do this is:

```
-- Apply an 'order by' filter to retrieve the COMPANY documents in ascending 'companyName'
order

SET OutputLocalEnvironment.Destination.Loopback.Request.filter.order[1].companyName = 'ASC';
```

Using ESQL to set an 'order' clause

The LocalEnvironment.Destination.Loopback.Request.filter.order folder can contain a list of fields whose values should be set to ASC to order in ascending order or DESC to order in descending order. In this case, we have chosen to set up an order by clause on just one field to retrieve the documents in ascending 'companyName' order.

The 'RetrieveCloudantDocuments' LoopBackRequest node is configured to retrieve the COMPANY documents from the CLOUDANT datasource, using the username and password from the CLOUDANT_SEC_ID 'Security identity' as we did our other Cloudant LoopBackRequest node.



Using the LoopBackRequest node to retrieve documents from the CLOUDANT datasource

We then wire this node to the HTTP Reply node to reply with the retrieved documents.

Using a 'LoopBack Model'

If we now test the 'Retrieve' part of the message flow with the Flow Exerciser things don't quite go as expected. The message fails to be processed and the 'RetrieveCloudantDocuments' LoopBackRequest node that uses the 'order' filter reports the following errors:

```
BIP2230E: ( TESTNODE_hattg.default ) Error detected whilst processing a message in node
'MongoRetrieveSyncCloudant.RetrieveCloudantDocuments'.
BIP3868E: ( TESTNODE_hattg.default ) The LoopBackRequest node received error code "Error'
' while performing the "Retrieve" operation. Error: "Unspecified or ambiguous sort type
. Try appending :number or :string to the sort field. companyName".
```

It seems that the Cloudant LoopBack connector needs to be provided with the data type of the 'companyName' field that we are trying to sort by and to tell the connector this information when using the LoopBackRequest node in IIB we need to define a LoopBack model. You can get more information on LoopBack models from the documentation but essentially a LoopBack model is a JSON file that represents data in a backend system. These model files can be created using a 'model generator' which is a tool provided by LoopBack or they can be handcrafted in an appropriate editor.

Models can be used to do things like set default values and enforce basic validation however, in this case, I simply used a text editor to create a basic model in a file called COMPANY.json which contains the following JSON data specifying just the properties and their data types:

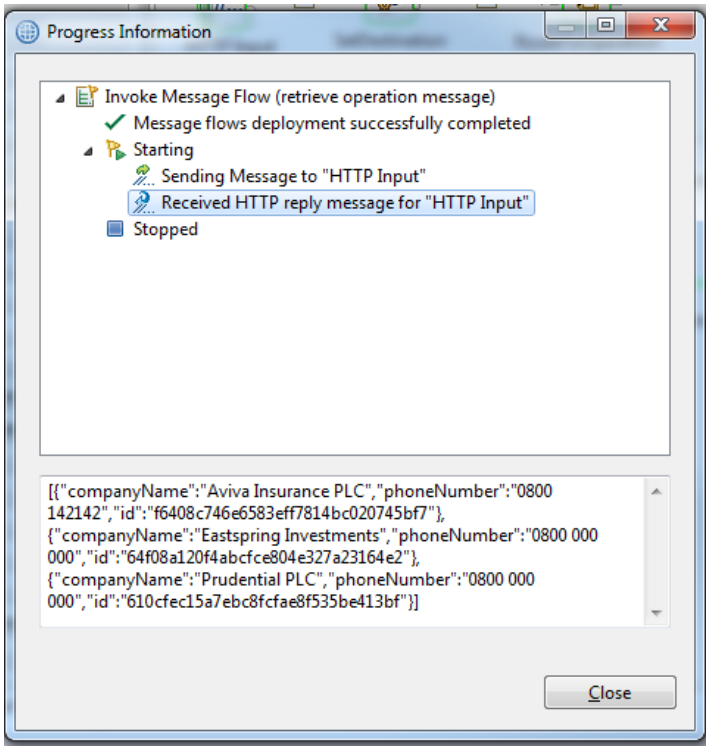
```
{
  "name": "COMPANY",
  "properties": {
    "companyName": {
      "type": "string"
    },
    "phoneNumber": {
      "type": "string"
    }
  }
}
```

A simple LoopBack 'model' for the COMPANY document

To use the model, the name of the file (excluding the .json suffix) must exactly match the value that is specified in the 'LoopBack object' property on the LoopBackRequest node or, if used, the LocalEnvironment.Destination.Loopback.Request.object override. This file must be placed into a subdirectory of the directory from which the datasources.json file containing the connection details for the datasource is loaded. This subdirectory must have the same name as the datasource being used for the connection.

So, in this example our datasource is called CLOUDANT and the datasources.json file that configures this datasource is located in C:\ProgramData\IBM\MQSI\connectors\loopback. To use this model file I therefore need to create the C:\ProgramData\IBM\MQSI\connectors\loopback\CLOUDANT directory and then copy COMPANY.json into it.

If we now try out the message flow again with the Flow Exerciser this time the message is processed to completion and the following output message is returned:



Retrieved COMPANY documents from Cloudant

Summary

In this post I have extended the simple message flow presented in Basic Introduction to Using the LoopBackRequest node to show how to use filters in the LocalEnvironment with the LoopBackRequest node when retrieving data from a MongoDB database. This has covered using

filters to specify:

- A ‘where’ clause to control which documents are returned.
- A ‘field’ filter to control which fields from those documents are retrieved.

I have also explained how to configure a datasource to connect to a Cloudant NoSQL database running as a service in IBM Bluemix so that the message flow could be enhanced to then create documents in a Cloudant database. We have also seen how to use a LoopBack model with the LoopBackRequest node when retrieving documents from a Cloudant database using an ‘order’ filter.

TAGS IBM-INTEGRATION-BUS, INTEGRATION, INTEGRATION BUS, FIX PACK V10.0.0.6, LOOPBACK

GREGH