IBM Integration Bus

A REST API processing a .jpeg image with the image included as a JSON message element

Featuring:

- REST API using JSON domain
- Handling embedded image encoded as Base64
- Testing using the Chrome Postman utility

November 2016

Hands-on lab built at product Version 10.0.0.6

1. Introduction

This lab guide is the seconds of two labs that show you how to create a REST API that processes a message that contains a binary image. In this case, the binary image is encoded as a BASE64 file, and the encoded data is transmitted as one of the elements in the REST JSON message.

This lab is based on the standard HR_Service example used in most of the labs in this series. However, to accommodate the encoded binary image, one additional element, IMAGE, has been added at the end of the EMPLOYEE model. Thus, the JSON model definition that is used in this lab is as follows (some sample data is shown).

```
"EMPLOYEE": {
     "EMPNO": "000003",
     "FIRSTNME": "Albert",
     "MIDINIT": "J",
     "LASTNAME": "Einstein",
     "WORKDEPT": "A00",
     "PHONENO": "0101",
     "HIREDATE": "1912-07-27",
     "JOB": "Manager",
     "EDLEVEL": 9,
     "SEX": "M",
     "BIRTHDATE": "1879-03-14",
     "SALARY": 9990,
     "BONUS": 4440,
     "COMM":6660,
     “IMAGE”: “xxxxxxxxxxxxxxxxxxxxxxxxxxx”
 }
```

xxxxxxxxx represents the binary data encoded as BASE64. In this lab, we have provided an image of Albert Einstein, and we have also provided the Base64 encoded version of this image. This was done with one of the many freeware utilities that are available on the internet.

1.1 Lab preparation

This lab is based on the solution of the REST API HR_Service. This lab uses an IIB node called TESTNODE_iibuser. You should have it already created. If not, refer to the REST API HR Service Lab Guide document.

This lab is based on a set of REST scenarios that are described in other lab guides in this series. Specifically, it may be useful to review the lab “Lab 02, Developing a REST API using a Swagger JSON document”.

This lab does not ask you to build the solution from scratch. A complete solution is provided, and you will investigate various aspects of the solution, and perform a test of the provided solution.

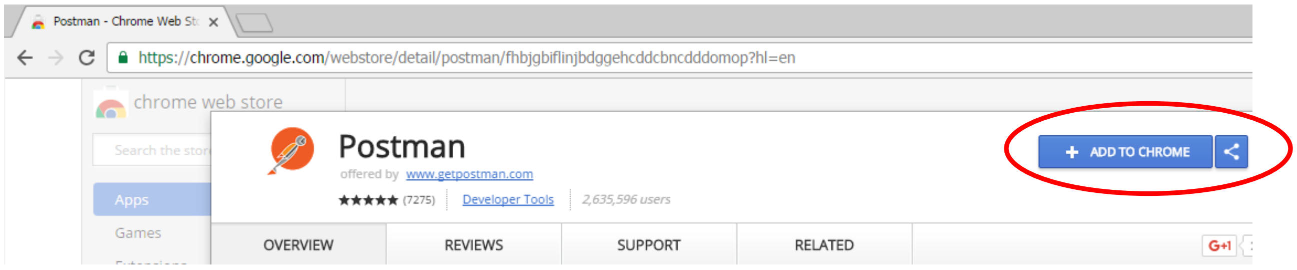
1.1.1 The Chrome Postman plugin

In this lab, we will use the Chrome Postman test tool to provide the client REST request. Alternative tools such as SOAPUI could also be used for this propose, but we have used Postman for consistency with the “REST with MIME attachment lab (lab 16L15), which requires Postman.

To obtain this tool, from a Chrome browser, search for “get Postman”. Select the “Postman Chrome Web Store” at Google, and click “Add to Chrome” to install.

Note that when it executes, this app does not actually run under Chrome; it executes as a stand-alone application.

If you are using the supplied VMWare workshop image, Postman has already been installed.



1.1.2 MQ Configuration

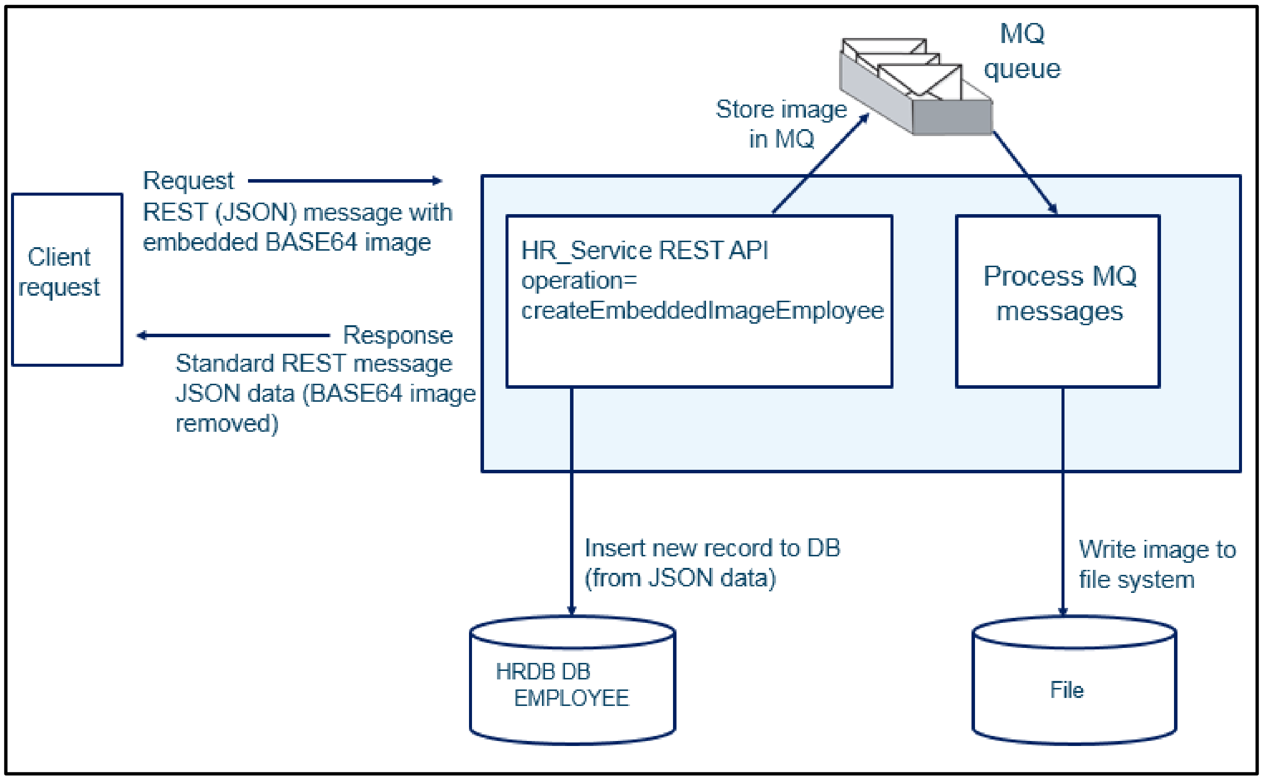
This lab uses an MQ queue to hold incoming images from the REST request prior to writing to a file.

In your queue manager (IB10QMGR on the workshop VMWare system), create an MQ queue called IMAGES.EMBEDDED.OUT.

1.2 Outline of Lab

This lab provides a technique for an IIB REST API to receive a REST request that contains an accompanying image.

In this case, the REST message contains a standard JSON-formatted message. The JSON message contains “new employee” data, and the IIB REST API will take this information and add a new row to the EMPLOYEE table. The JSON message also contains an additional element, IMAGE, which contains a Base64 encoded version of the related image (a .jpeg file). This image is processed by the REST API, and is written to a local MQ queue for subsequent processing.


MQ queue
Store image in MQ
Request
REST (JSON) message with embedded BASE64 image
Client request
HR_Service REST API operation= createEmbeddedImageEmployee
Process MQ messages
Response
Standard REST message JSON data (BASE64 image removed)
Insert new record to DB (from JSON data)
Write image to file system
HRDB DB EMPLOYEE
File

1.3 Configure TESTNODE_iibuser for REST APIs

The instructions in this lab guide are based on a Windows implementation, with a user named "iibuser".
The Windows VMWare image on which this lab is based is not available outside IBM, so you will need to provide your own software product installations where necessary.

Login to Windows as the user "iibuser", password = "passw0rd". (You may already be logged in).

Start the IIB Toolkit from the Start menu.

The IIB support for the REST API requires some special configuration for the IIB node and server.

If you have already done a previous lab involving the REST API function in this series of lab guides, you can skip to the next heading.

The IIB support for the REST API requires some special configuration for the IIB node and server.

1. Ensure that TESTNODE_iibuser is started.

2. Enable Cross-Origin Resource Scripting for REST. This is required when testing with the SwaggerUI test tool. See http://www.w3.org/TR/cors/?cm_mc_uid=09173639950214518562833&cm_mc_sid_50200000=1452177651 for further information.

   (Helpful hint - the VM keyboard is set to UK English. If you cannot find the "\" with your keyboard, use "cd .." to move the a higher-level folder in a DOS window), or change the keyboard settings to reflect your locale.)

   In an IIB Integration Console (shortcut on the Start menu), run the following command.

   Note, the text should be typed on a single line - the parameters are shown on different lines here for readability; the same approach is taken throughout this and other lab documents.

```
mqsichangeproperties TESTNODE_iibuser
      -e default
      -o HTTPConnector
      -n corsEnabled -v true
```

1.4 Configure Integration Bus node to work with DB2

If you have already done a previous lab involving the HRDB database in this series of lab guides, you can skip to the next heading.

To run this lab, the Integration Bus node must be enabled to allow a JDBC connection to the HRDB database.

1. Open an IIB Command Console (from the Start menu), and navigate to

   ```
   c:\student10\Create_HR_database
   ```

2. Run the command

   ```
   3_Create_JDBC_for_HRDB
   ```

   Accept the defaults presented in the script. This will create the required JDBC configurable service for the HRDB database.

3. Run the command

   ```
   4_Create_HRDB_SecurityID
   ```

4. Stop and restart the node to enable the above definitions to be activated

   ```
   mqsistop TESTNODE_iibuser

   mqsistart TESTNODE_iibuser
   ```

This will create the necessary security credentials enabling TESTNODE_iibuser to connect to the database.

Recreating the HRDB database and tables
The HRDB database, and the EMPLOYEE and DEPARTMENT tables have already been created on the supplied VMWare image. If you wish to recreate your own instance of this database, the command **`1_Create_HRDB_database.cmd`** and **`2_Create_HRDB_Tables.cmd`** are provided for this. If used in conjunction with the VM image, these commands must be run under the user "iibadmin". Appropriate database permissions are included in the scripts to GRANT access to the user iibuser.

2. Import the REST API

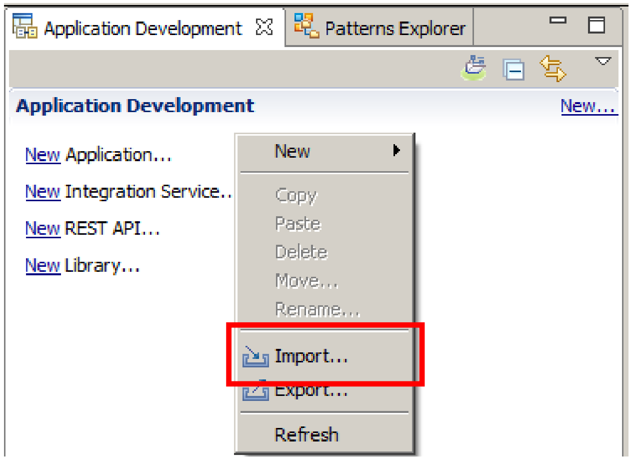
In this part of the lab exercise, you will import and deploy the provided IIB shared library and REST API.

2.1 Import the IIB REST API

To avoid naming clashes with earlier labs, this lab will be developed using a new workspace.

In the Integration Toolkit, click File, Switch Workspace. Give the new workspace the name "embedded_image", or similar.

Right-click in the Application Development pane and click 'Import':



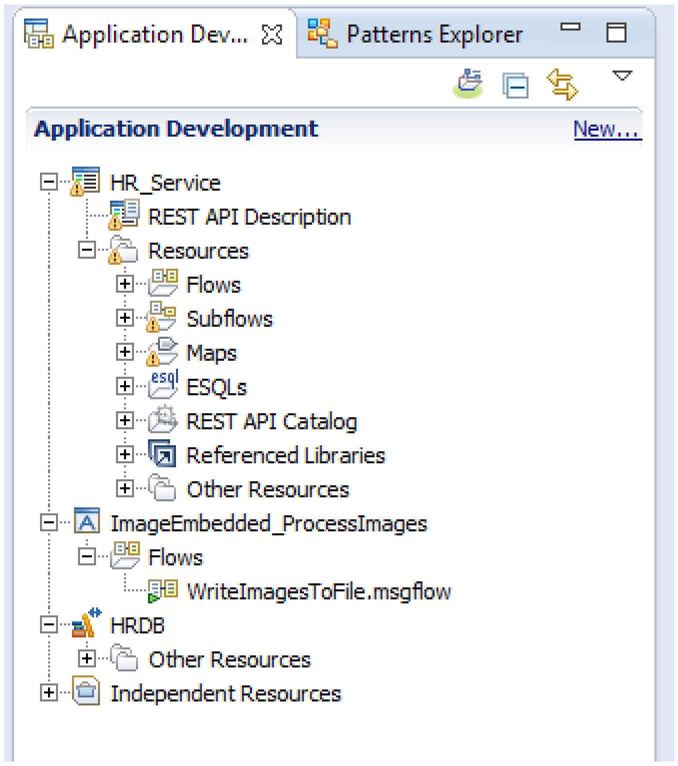
Import the following Project Interchange (PI) file:

C:\student10\REST_withImages\REST_embeddedImage\solution\ HR_Service_embeddedImage.10.0.0.6.zip

Note: Make sure that all projects in this PI file are selected for import. The PI includes the HRDB shared library and database project.

When imported, you should have in your workspace the **HR_Service** REST API and the HRDB shared library that is referenced by the REST API.

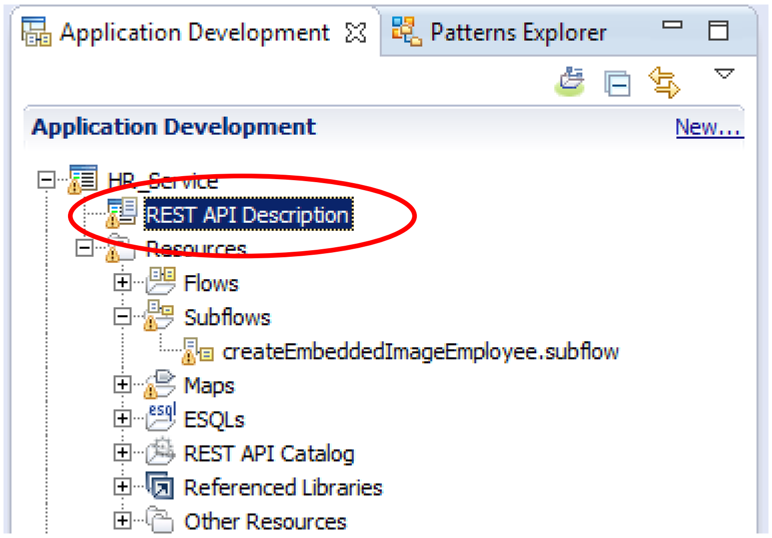
The application **ImageEmbedded_ProcessImages** is also provided to perform the subsequent processing of images using MQ. This will be described towards the end of this lab.



2.2 Explore the HR_Service REST API

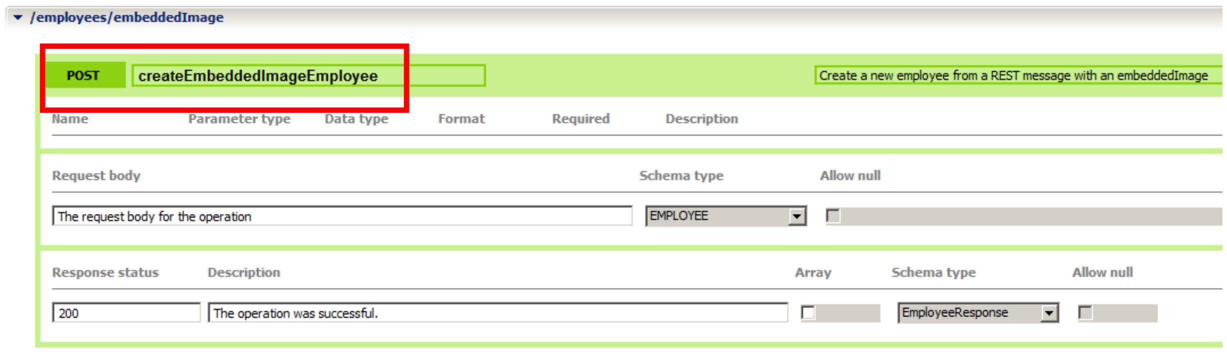
In this section, you will explore the imported REST API.

1. Expand the REST API Service and double-click “REST API Description”.



2. You will be familiar with the REST API Description, Header and Resources from earlier lab scenarios, so proceed to the Resources section, and focus on the **createEmbeddedImageEmployee** POST operation in the /**employees/embeddedimage** resource.

   Note that the Request body has a schema type of EMPLOYEE, and the Response body has a schema type of EmployeeResponse.



/employees/embeddedImage

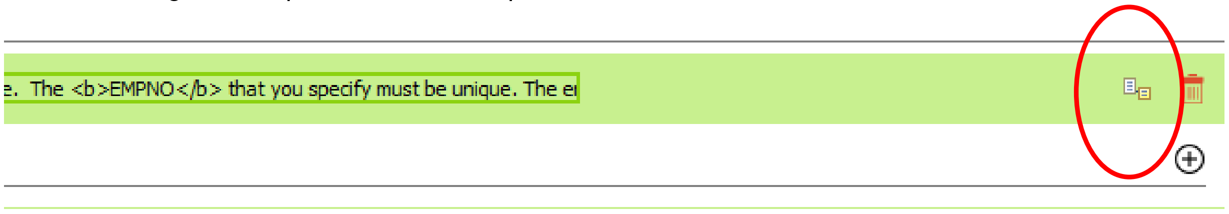
POST createEmbeddedImageEmployee — Create a new employee from a REST message with an embeddedImage

Name	Parameter type	Data type	Format	Required	Description

Request body	Schema type	Allow null
The request body for the operation	EMPLOYEE	☐

Response status	Description	Array	Schema type	Allow null
200	The operation was successful.	☐	EmployeeResponse	☐

3. Scroll to the right, and open the subflow implementation.



e. The <b>EMPNO</b> that you specify must be unique. The en

4. This opens the implementation subflow. Each node will be investigated in detail over the next few pages.

   At a high level, the subflow performs the following actions:

   1. The first mapping node inserts the new employee row into the HRDB database, using the JSON data from the message tree as input. It also extracts the encoded base42 image (the IMAGE element) and invokes a small custom ESQL transform to decode the Base64 image. This is then saved in the Environment tree.

      The ESQL module is called decodeBase64Image.esql.

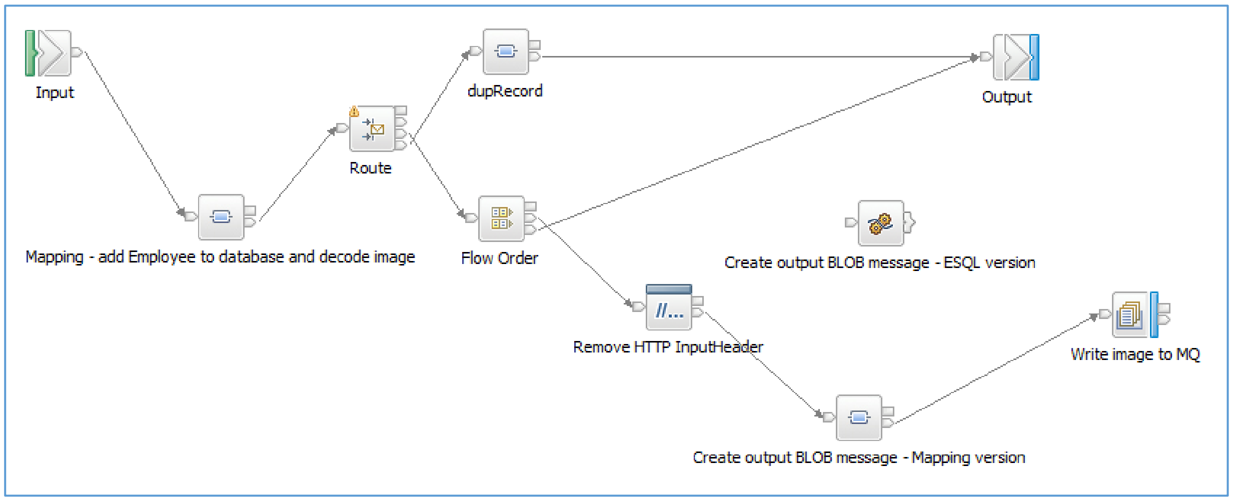
      Note that the IMAGE element is then discarded, and not included in the message tree.

   2. The Route node tests whether the insert was successful.
      - If the insert was successful (rows inserted not = 0), then control passes to the FlowOrder node
      - If the insert was not successful, a duplicate key may have been detected (control passes to the dupRecord map node). Otherwise, control passes directly to the response Compute node.

   3. The FlowOrder node will:
      1. Send the message tree to the “Create output BLOB message” compute node for processing of the binary image part.
      2. Send a response to the client (userReturnCode=0 will indicate success to the client).

   4. The “Create output BLOB message” node retrieves the binary image from the Environment tree, and reinstates it in the main message tree, from where it is subsequently written to a holding MQ queue, for later processing. This approach was taken because it allows the employee image to be stored transactionally (ie. to MQ), before being written to a file by a subsequent application.

      Two versions of this function are provided – a map node and an ESQL compute node.

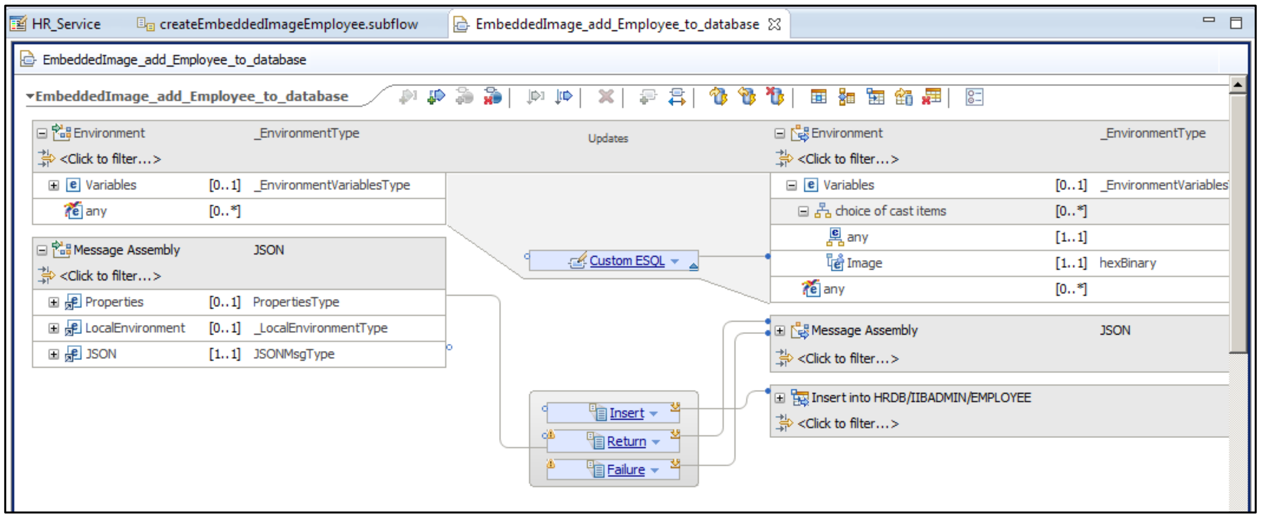

Input
dupRecord
Output
Route
Mapping - add Employee to database and decode image
Flow Order
Create output BLOB message - ESQL version
Remove HTTP InputHeader
Write image to MQ
Create output BLOB message - Mapping version

2.3 Explore the REST API in detail

1. Open the map “Mapping – add Employee to database and decode image”.

   This map has an input assembly in JSON format, which contains a full Employee record. The map inserts the EMPLOYEE data into the HRDB/EMPLOYEE table.

   Failure scenarios such as “duplicate record” are handled by saving database returned data such as SQLSTATE in the output message.



HR_Service
createEmbeddedImageEmployee.subflow
EmbeddedImage_add_Employee_to_database
EmbeddedImage_add_Employee_to_database
EmbeddedImage_add_Employee_to_database
Updates
Custom ESQL
Insert
Return
Failure

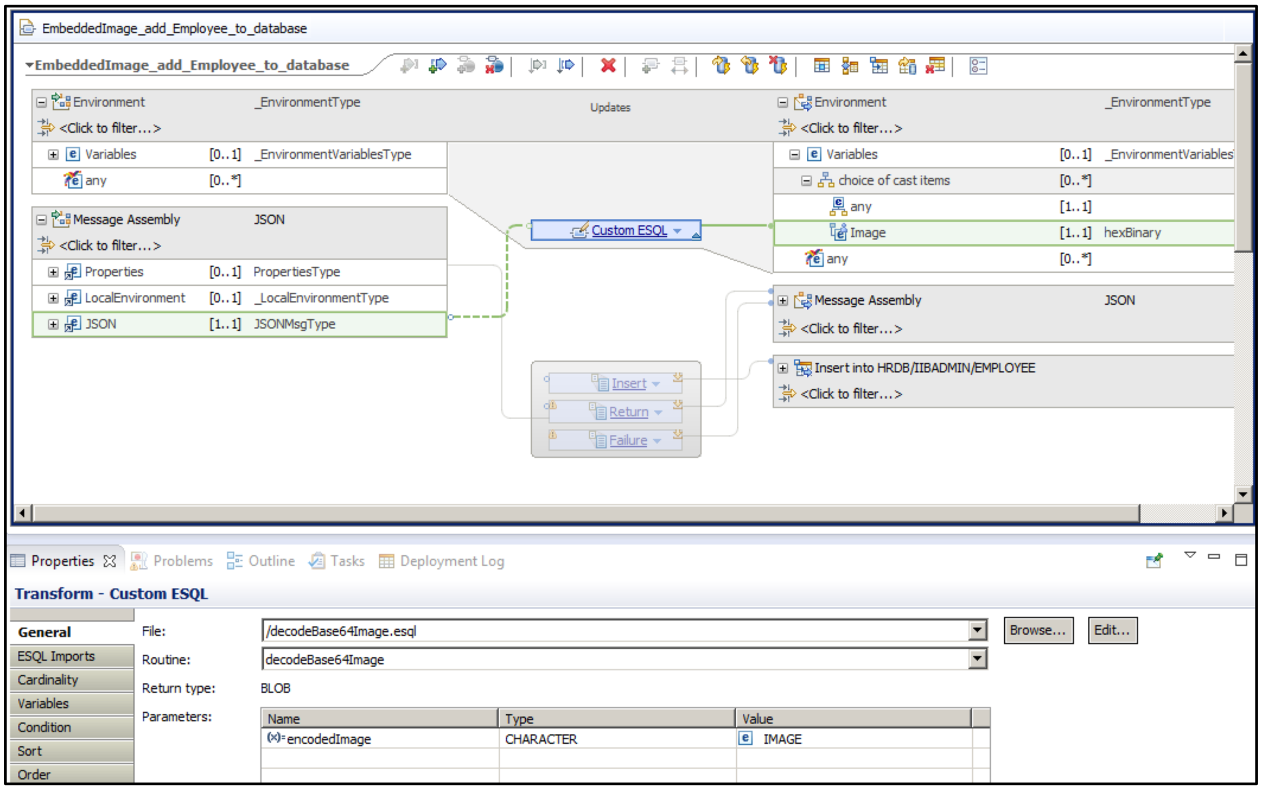
Environment		_EnvironmentType
<Click to filter...>		
Variables	[0..1]	_EnvironmentVariablesType
any	[0..*]	
Message Assembly		JSON
<Click to filter...>		
Properties	[0..1]	PropertiesType
LocalEnvironment	[0..1]	_LocalEnvironmentType
JSON	[1..1]	JSONMsgType

Environment		_EnvironmentType
<Click to filter...>		
Variables	[0..1]	_EnvironmentVariables
choice of cast items	[0..*]	
any	[1..1]	
Image	[1..1]	hexBinary
any	[0..*]	
Message Assembly		JSON
<Click to filter...>		
Insert into HRDB/IIBADMIN/EMPLOYEE		
<Click to filter...>		

2. Highlight the Custom ESQ transform by clicking it.

   In the Properties of the transform, note that parameter encodedImage has been created. The Type is CHARACTER, and the value is IMAGE. This means that the input IMAGE element is sent by the map to the ESQL module input parameter encodedImage.

   The output of the transform is connected to the output element Environment/Variables/Image. This is defined as hexBinary, and holds the decoded version of the IMAGE element.



Transform - Custom ESQL

File: /decodeBase64Image.esql

Routine: decodeBase64Image

Return type: BLOB

Parameters:

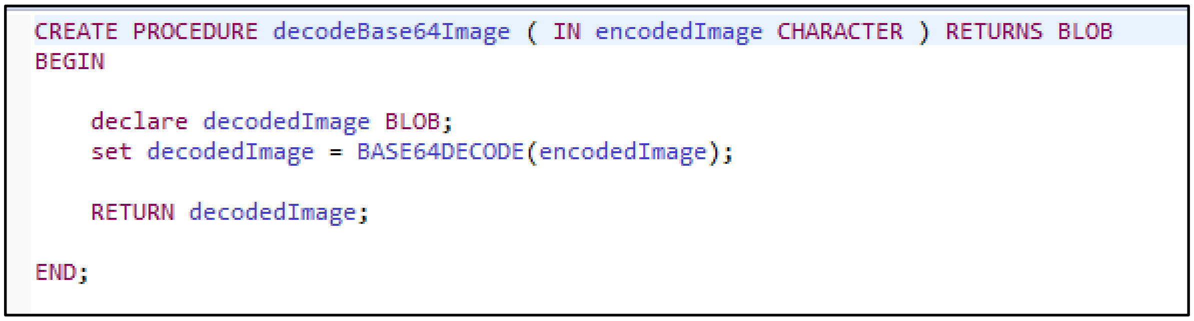
Name	Type	Value
encodedImage	CHARACTER	IMAGE

3. Open the ESQL module decodeBase64Image.esql.

   The ESQL is created as a Procedure, and the input parameter to the procedure is **encodedImage** (as mentioned in the corresponding map ESQL transform).

   The ESQL command BASE64DECODE converts an element that is held as BASE64 encoded, and decodes it into the corresponding binary element.

   The converted image is returned to the map by specifying the **RETURN** statement with the “**decodedImage**” parameter.

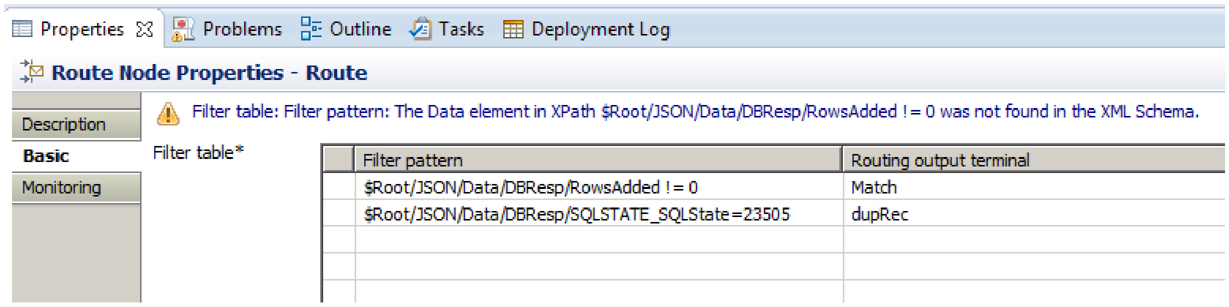


```
CREATE PROCEDURE decodeBase64Image ( IN encodedImage CHARACTER ) RETURNS BLOB
BEGIN

    declare decodedImage BLOB;
    set decodedImage = BASE64DECODE(encodedImage);

    RETURN decodedImage;

END;
```

4. Highlight the Route node and review the node properties.

   - The Match terminal is used when the database insert was successful (when the number of rows added was not zero).
   - The dupRec terminal is used when the SQLSTATE value is 23505 (SQL duplicate row). You can extend the flow yourself if you want to check for other specific returns.

   You will see a message suggesting that the Data element in the Filter pattern was not found in the XML schema. This is because the XPath Expression builder does not support the JSON form of messages, so XPath evaluation expressions (including filter patterns) must be manually built.



Route Node Properties - Route

Filter table: Filter pattern: The Data element in XPath $Root/JSON/Data/DBResp/RowsAdded != 0 was not found in the XML Schema.

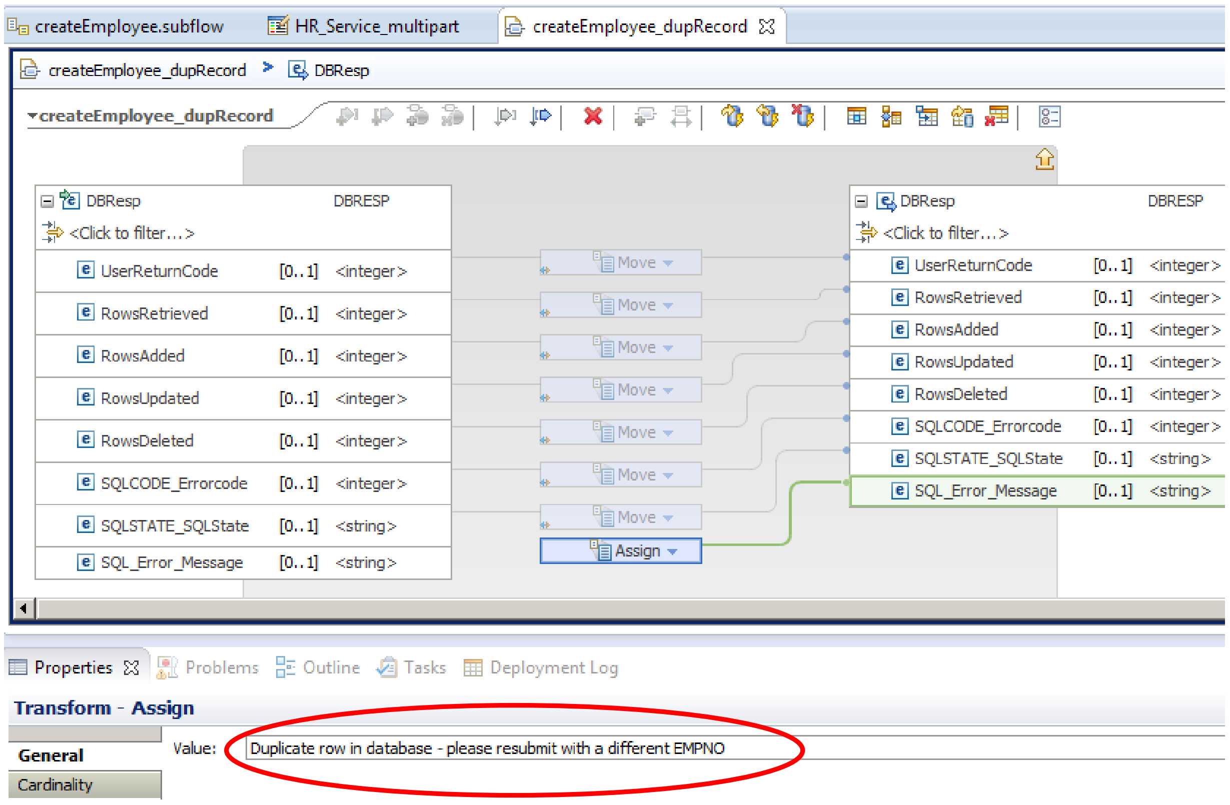
Filter table*

Filter pattern	Routing output terminal
$Root/JSON/Data/DBResp/RowsAdded != 0	Match
$Root/JSON/Data/DBResp/SQLSTATE_SQLState=23505	dupRec

5. Open the dupRecord record map node. Click “Local Map” in the main map editor.

   Review the various mappings that are provided. In particular, the SQL_Error_Message is set with an Assign transform, setting the message to a more readable form of the SQL error message.

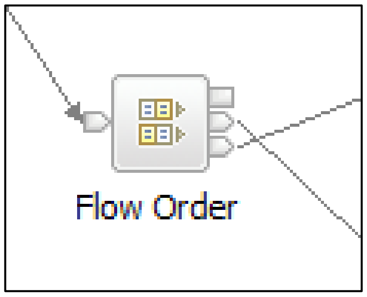
   Close the map.


createEmployee.subflow
HR_Service_multipart
createEmployee_dupRecord
createEmployee_dupRecord > DBResp
createEmployee_dupRecord
DBResp DBRESP
<Click to filter...>
UserReturnCode [0..1] <integer>
RowsRetrieved [0..1] <integer>
RowsAdded [0..1] <integer>
RowsUpdated [0..1] <integer>
RowsDeleted [0..1] <integer>
SQLCODE_Errorcode [0..1] <integer>
SQLSTATE_SQLState [0..1] <string>
SQL_Error_Message [0..1] <string>
Move
Assign
Properties
Problems
Outline
Tasks
Deployment Log
General
Cardinality

Transform - Assign

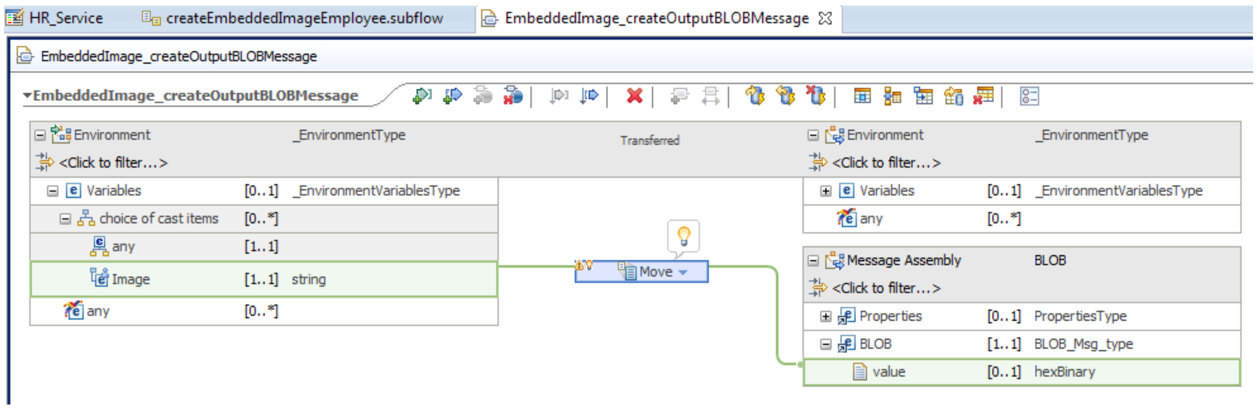
Value: Duplicate row in database - please resubmit with a different EMPNO

6. The Flow Order node first invokes the processing of the image contained in the original request. Secondly, it sends a response to the invoking client.



7. Open the **"Create output BLOB message"** map.

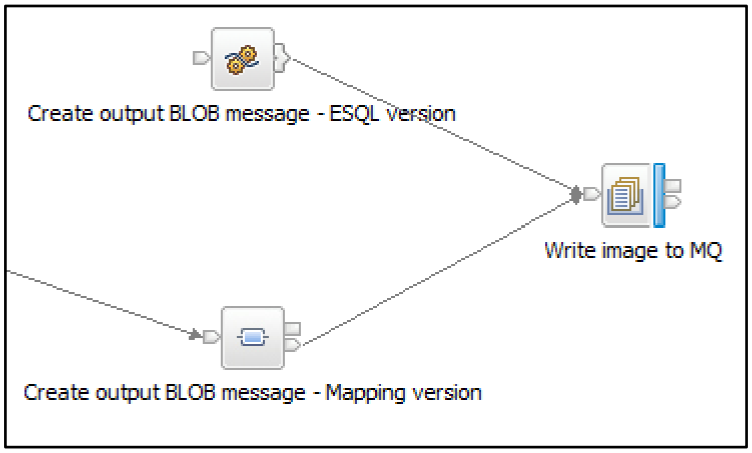
   The map performs a simple copy (Move) of the **Environment/Variables/Image** element to the output **BLOB/value** element (still defined as hexBinary).


HR_Service
createEmbeddedImageEmployee.subflow
EmbeddedImage_createOutputBLOBMessage
EmbeddedImage_createOutputBLOBMessage
EmbeddedImage_createOutputBLOBMessage
Transferred
Move

Environment		_EnvironmentType
<Click to filter...>		
Variables	[0..1]	_EnvironmentVariablesType
choice of cast items	[0..*]	
any	[1..1]	
Image	[1..1]	string
any	[0..*]	

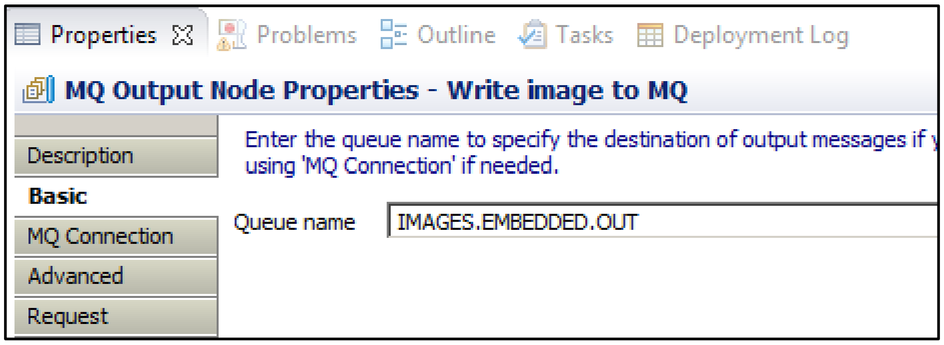
Environment		_EnvironmentType
<Click to filter...>		
Variables	[0..1]	_EnvironmentVariablesType
any	[0..*]	
Message Assembly		BLOB
<Click to filter...>		
Properties	[0..1]	PropertiesType
BLOB	[1..1]	BLOB_Msg_type
value	[0..1]	hexBinary

8. The remainder of the flow sends the output MQ message. An MQ Output node is provided to write the message to a local queue, IMAGES.EMBEDDED.OUT.


Create output BLOB message - ESQL version
Write image to MQ
Create output BLOB message - Mapping version

9. Click the MQOutput node to highlight its properties.

   On the Basic tab, note that the Queue Name has been set to **IMAGES.EMBEDDED.OUT**. If you are running this lab on your own system, you will need to define this queue on your nominated MQ queue manager.



MQ Output Node Properties - Write image to MQ

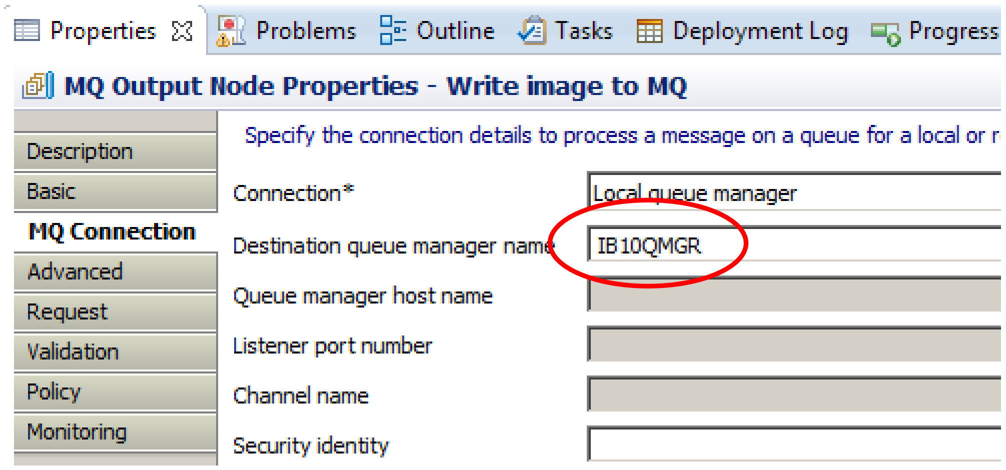
Enter the queue name to specify the destination of output messages if y
using 'MQ Connection' if needed.

Queue name: IMAGES.EMBEDDED.OUT

10. On the MQ Connection tab, note that the Destination queue manager name has been set to IB10QMGR.

    As in the previous step, if you are running this lab on your own system, you will need to change this property to the name of your own queue manager.

    If required on your own system, you may also need to change the Connection type to a value other than “Local queue manager”.



MQ Output Node Properties - Write image to MQ

Specify the connection details to process a message on a queue for a local or r

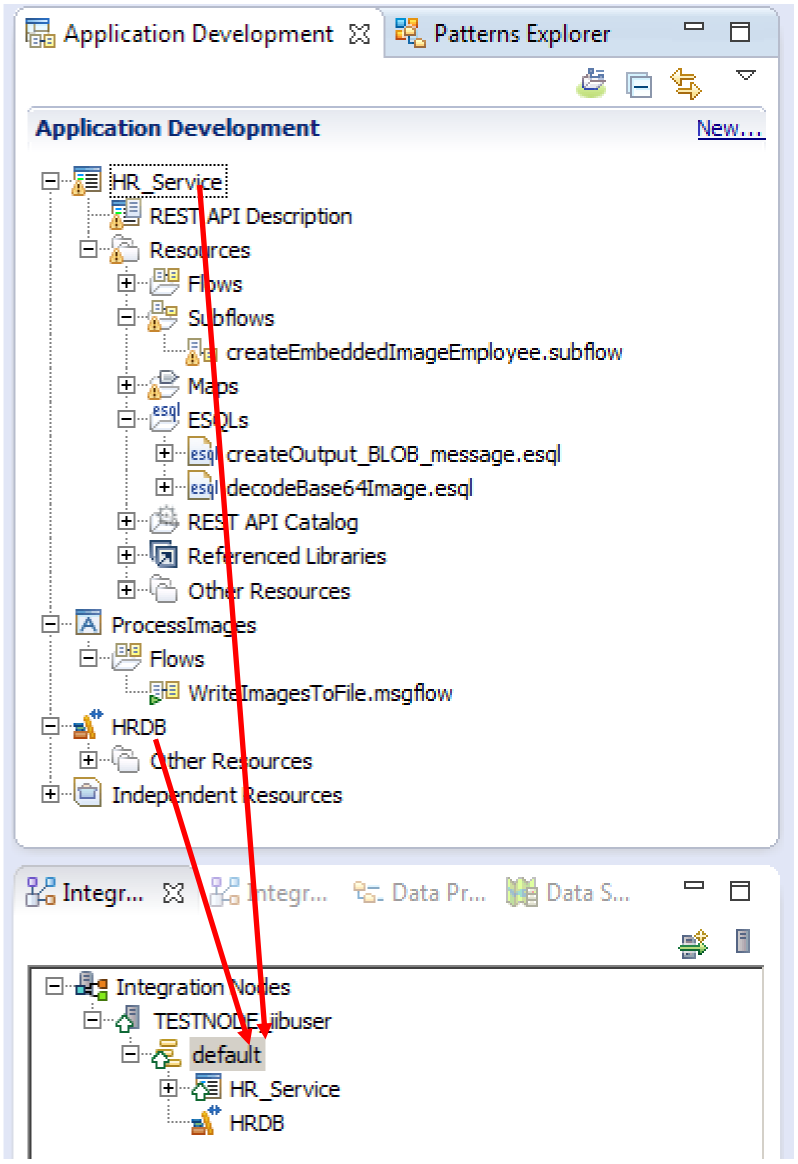
Connection*: Local queue manager
Destination queue manager name: IB10QMGR
Queue manager host name:
Listener port number:
Channel name:
Security identity:

3. Deploy and Test the REST API

3.1 Deploy

1. Deploy the following resources:
   - HRDB shared library
   - HR_Service

   Do not deploy the ProcessImages application at this time.



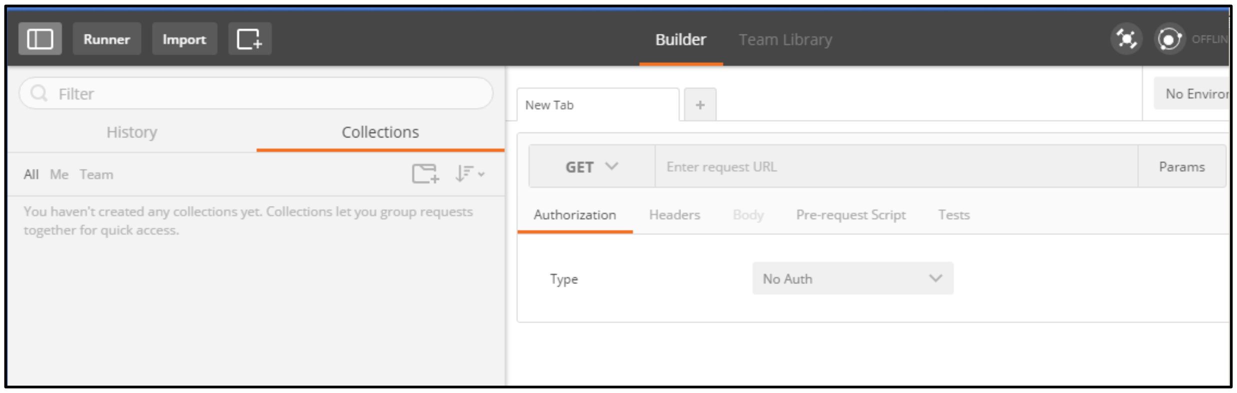
3.2 Test

3.2.1 Start Postman test tool

1. From the Start menu, or from your installation folder, start the Postman tool (in the workshop VM system, type Postman into the Start Search menu).

   After the progress message…

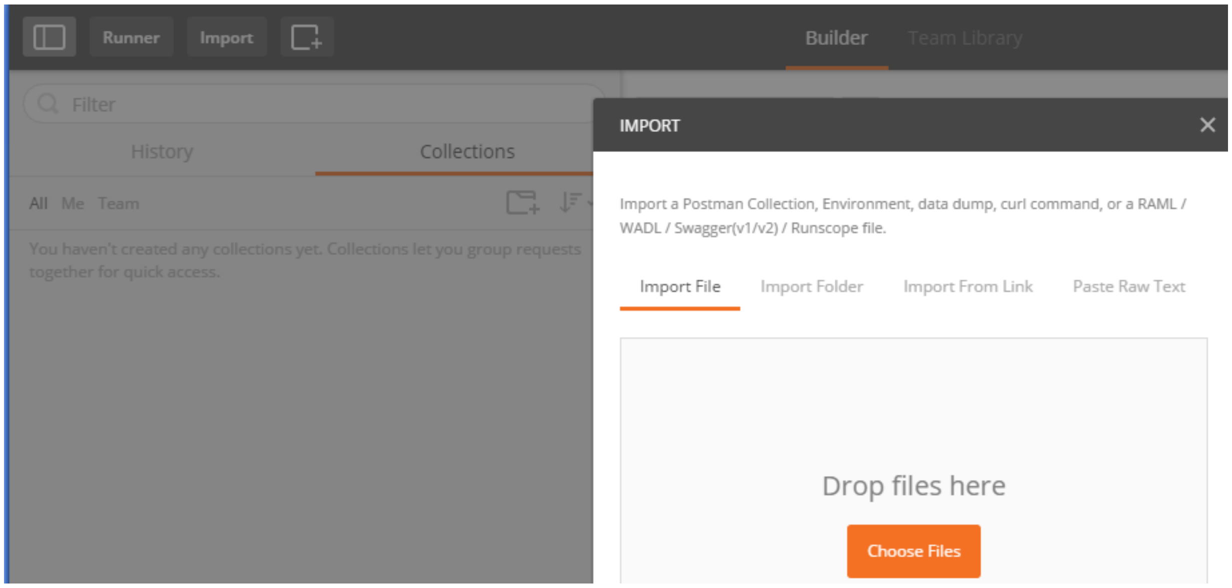
   … you will see the Postman main menu.



You haven't created any collections yet. Collections let you group requests together for quick access.

2. Import the Postman project for the EmbeddedImage lab. Click Import and import the file:

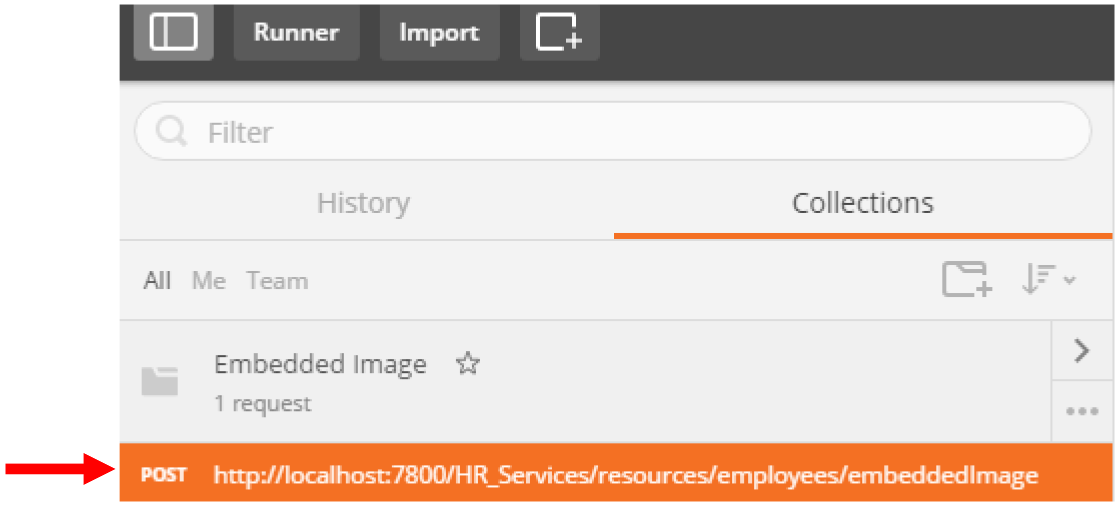
`c:\student10\REST_withImages\REST_embeddedImage\postman\EmbeddedImage.postman_collection.json`



You haven't created any collections yet. Collections let you group requests together for quick access.

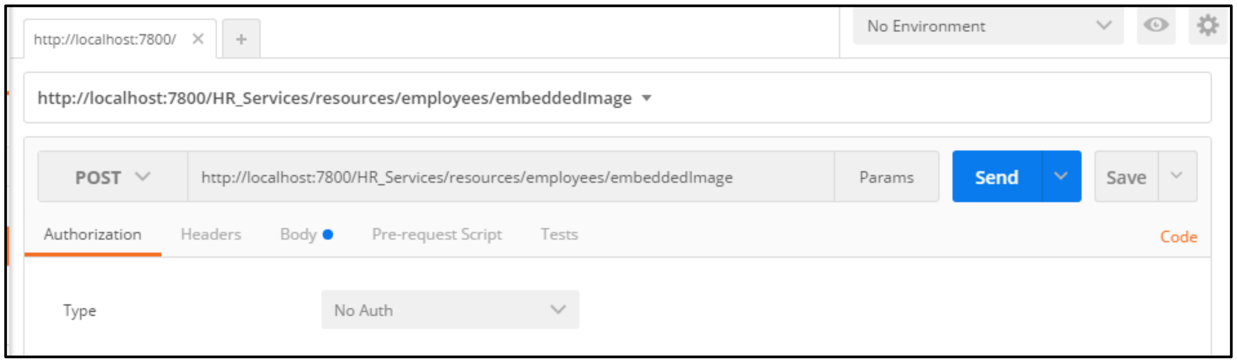
Import a Postman Collection, Environment, data dump, curl command, or a RAML / WADL / Swagger(v1/v2) / Runscope file.

3. Highlight the line with the POST request.



POST http://localhost:7800/HR_Services/resources/employees/embeddedImage

4. On the right pane, note that the URL is set to the required URL for the imported REST API. If your hostname or port are different, you will need to make appropriate changes to these values.



http://localhost:7800/HR_Services/resources/employees/embeddedImage

POST http://localhost:7800/HR_Services/resources/employees/embeddedImage

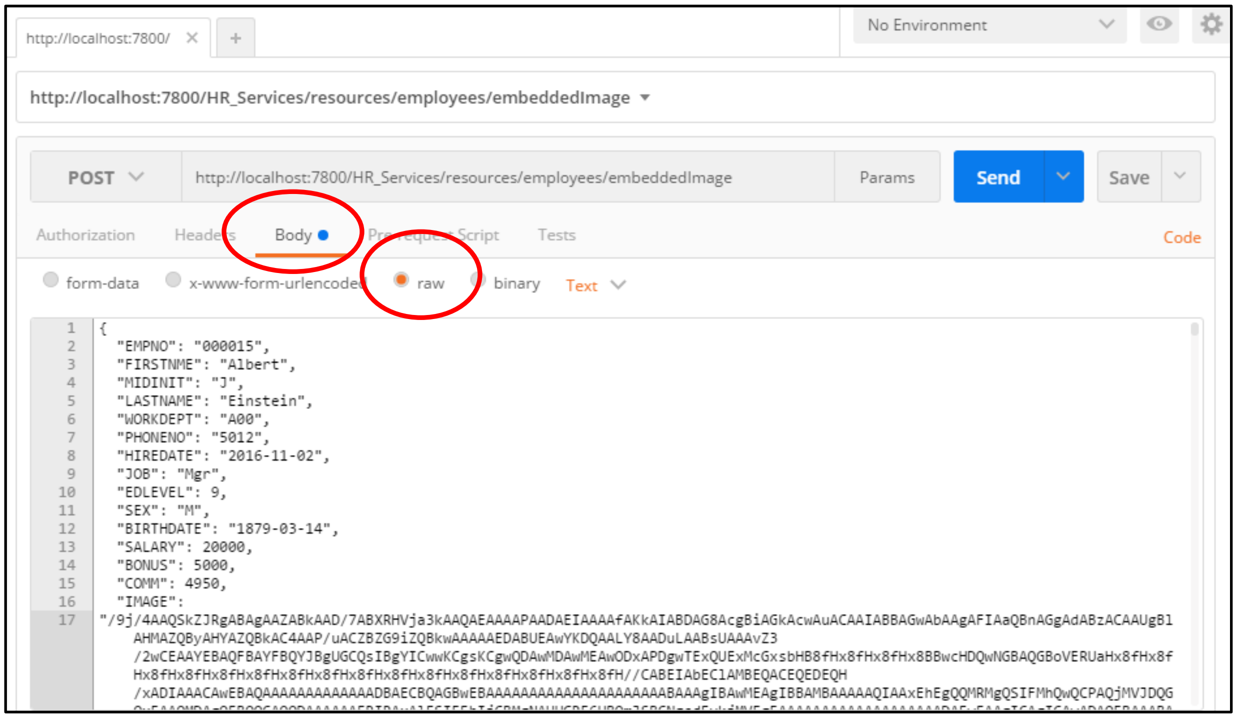
Type: No Auth

5. Click the “Body” tab.

   Click the “raw” form of the message (form-data should not be used for this type of message).

   Note that all the familiar element names of the EMPLOYEE message are present. In addition, the new element IMAGE is shown at the end of the element list. The content of the IMAGE element is the base64 encoded version of the binary image.

   For backup, this data is also stored in the file
   `C:\student10\REST_withImages\REST_embeddedImage\data\Einstein_Base64encoded_completeJSONmessage.txt`
   and can be copied/pasted into Postman if required.



http://localhost:7800/HR_Services/resources/employees/embeddedImage

POST http://localhost:7800/HR_Services/resources/employees/embeddedImage

○ form-data ○ x-www-form-urlencoded [✓ raw] ○ binary Text

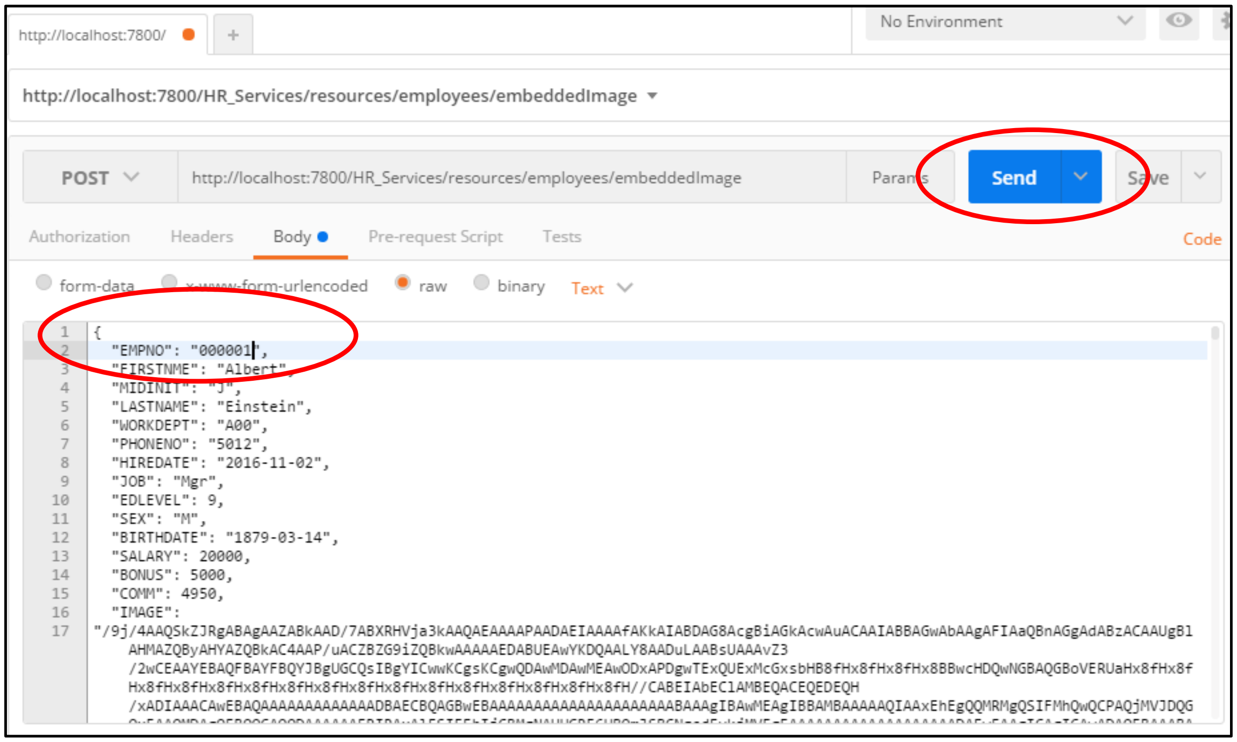
```
{
  "EMPNO": "000015",
  "FIRSTNME": "Albert",
  "MIDINIT": "J",
  "LASTNAME": "Einstein",
  "WORKDEPT": "A00",
  "PHONENO": "5012",
  "HIREDATE": "2016-11-02",
  "JOB": "Mgr",
  "EDLEVEL": 9,
  "SEX": "M",
  "BIRTHDATE": "1879-03-14",
  "SALARY": 20000,
  "BONUS": 5000,
  "COMM": 4950,
  "IMAGE":
```

"/9j/4AAQSkZJRgABAgAAZABkAAD/7ABXRHVja3kAAQAEAAAAPAADAEIAAAAfAKkAIABDAG8AcgBiAGkAcwAuACAAIABBAGwAbAAgAFIAaQBnAGgAdABzACAAUgBl
AHMAZQByAHYAZQBkAC4AAP/uACZBZG9iZQBkwAAAAAEDABUEAwYKDQAALY8AADuLAABsUAAAvZ3
/2wCEAAYEBAQFBAYFBQYJBgUGCQsIBgYICwwKCgsKCgwQDAwMDAwMEAwODxAPDgwTExQUExMcGxsbHB8fHx8fHx8fHx8BBwcHDQwNGBAQGBoVERUaHx8fHx8f
Hx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fH//CABEIAbEClAMBEQACEQEDEQH
/xADIAAACAwEBAQAAAAAAAAAAAAAADBAECBQAGBwEBAAAAAAAAAAAAAAAAAAAABAAAgIBAwMEAgIBBAMBAAAAAQIAAxEhEgQQMRMgQSIFMhQwQCPAQjMVJDQG

3.2.2 Test with Postman

1. Set the employee number (EMPNO) to one that is known not to already exist (eg. 000001).

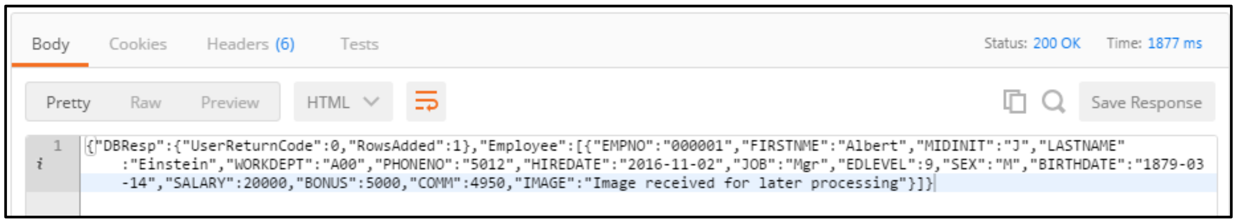
   Click Send.



```
{
  "EMPNO": "000001",
  "FIRSTNME": "Albert",
  "MIDINIT": "J",
  "LASTNAME": "Einstein",
  "WORKDEPT": "A00",
  "PHONENO": "5012",
  "HIREDATE": "2016-11-02",
  "JOB": "Mgr",
  "EDLEVEL": 9,
  "SEX": "M",
  "BIRTHDATE": "1879-03-14",
  "SALARY": 20000,
  "BONUS": 5000,
  "COMM": 4950,
  "IMAGE":
"/9j/4AAQSkZJRgABAgAAZABkAAD/7ABXRHVja3kAAQAEAAAAPAADAEIAAAAfAKkAIABDAG8AcgBiAGkAcwAuACAAIABBAGwAbAAgAFIAaQBnAGgAdABzACAAUgBl
    AHMAZQByAHYAZQBkAC4AAP/uACZBZG9iZQBkwAAAAAEDABUEAwYKDQAALY8AADuLAABsUAAAvZ3
    /2wCEAAYEBAQFBAYFBQYJBgUGCQsIBgYICwwKCgsKCgwQDAwMDAwMEAwODxAPDgwTExQUExMcGxsbHB8fHx8fHx8fHx8BBwcHDQwNGBAQGBoVERUaHx8fHx8f
    Hx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fHx8fH//CABEIAbEC1AMBEQACEQEDEQH
    /xADIAAACAwEBAQAAAAAAAAAAAAAADBAECBQAGBwEBAAAAAAAAAAAAAAAAAAABAAAgIBAwMEAgIBBBAMBAAAAAQIAAxEhEgQQMRMgQSIFMhQwQCPAQjMVJDQG
```

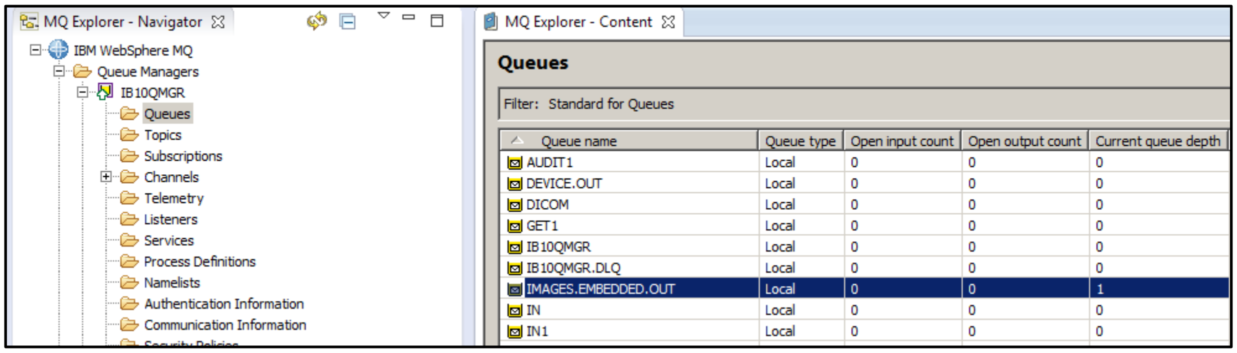
2. The input employee data will be returned, along with userReturnCode=0, indicating that the data was successfully added to the database.

   Note that the IMAGE element has not been returned with the original data, but now contains the message “Image received for later processing”.



```
{"DBResp":{"UserReturnCode":0,"RowsAdded":1},"Employee":[{"EMPNO":"000001","FIRSTNME":"Albert","MIDINIT":"J","LASTNAME"
    :"Einstein","WORKDEPT":"A00","PHONENO":"5012","HIREDATE":"2016-11-02","JOB":"Mgr","EDLEVEL":9,"SEX":"M","BIRTHDATE":"1879-03
    -14","SALARY":20000,"BONUS":5000,"COMM":4950,"IMAGE":"Image received for later processing"}]}
```

3. Using MQ Explorer, or your MQ tool of choice, observe that the queue IMAGES.EMBEDDED.OUT has one message.



Queues

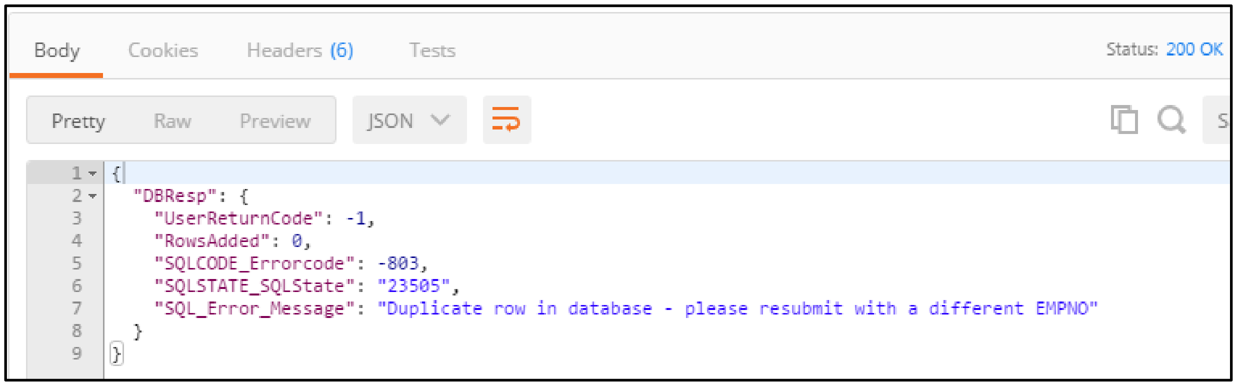
Filter: Standard for Queues

Queue name	Queue type	Open input count	Open output count	Current queue depth
AUDIT1	Local	0	0	0
DEVICE.OUT	Local	0	0	0
DICOM	Local	0	0	0
GET1	Local	0	0	0
IB10QMGR	Local	0	0	0
IB10QMGR.DLQ	Local	0	0	0
IMAGES.EMBEDDED.OUT	Local	0	0	1
IN	Local	0	0	0
IN1	Local	0	0	0

4. Return to the Postman tool, and click Send again, without changing the value of EMPNO.

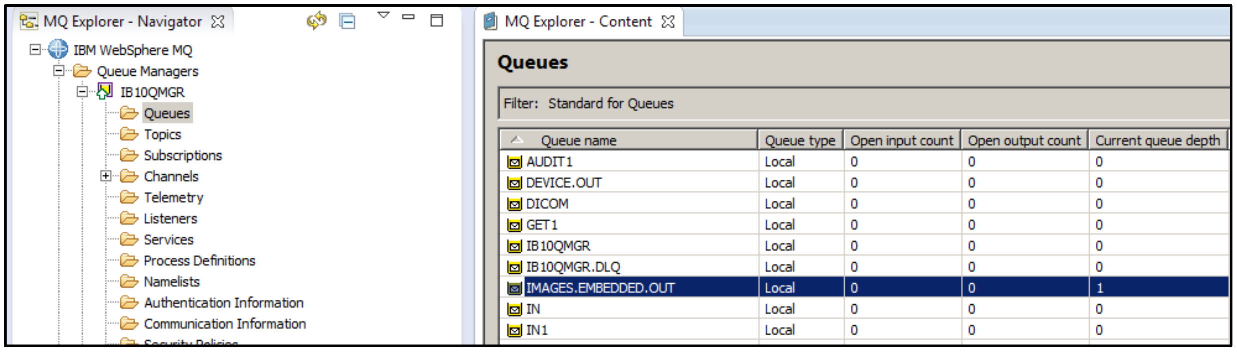
   Note that this time, the response shows that you are trying to add a duplicate row to the database, and this has been rejected.

   Note – this response is only received if the database EMPLOYEE table definition has defined the EMPNO column as UNIQUE, or has defined EMPNO as a primary key. The HRDB table definition DDL (in c:student10\Create_HR_database\Create_HRDB_Tables1.ddl) has been defined with a primary key in this way.



```
{
  "DBResp": {
    "UserReturnCode": -1,
    "RowsAdded": 0,
    "SQLCODE_Errorcode": -803,
    "SQLSTATE_SQLState": "23505",
    "SQL_Error_Message": "Duplicate row in database - please resubmit with a different EMPNO"
  }
}
```

5. Back in MQ Explorer, double-check that the number of MQ messages waiting to be processed is still 1. This means that the incoming message (attached image) has not been written to the queue manager.



Queues

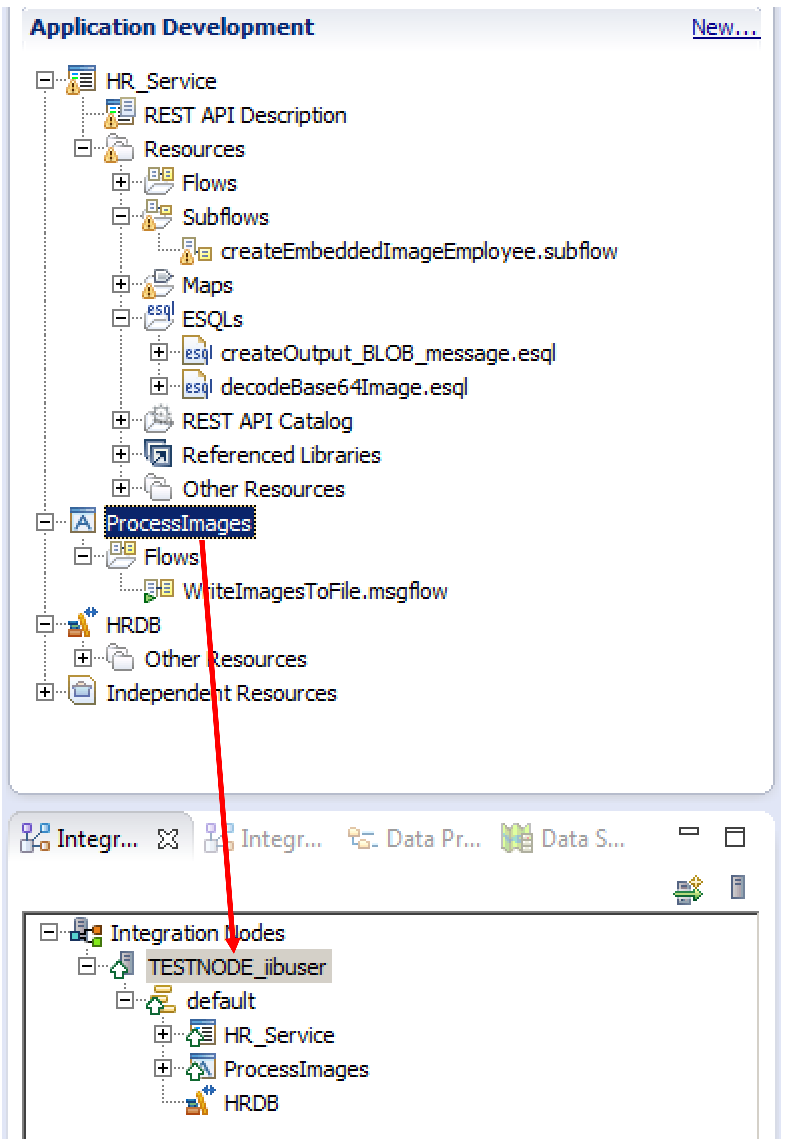
Filter: Standard for Queues

Queue name	Queue type	Open input count	Open output count	Current queue depth
AUDIT1	Local	0	0	0
DEVICE.OUT	Local	0	0	0
DICOM	Local	0	0	0
GET1	Local	0	0	0
IB10QMGR	Local	0	0	0
IB10QMGR.DLQ	Local	0	0	0
IMAGES.EMBEDDED.OUT	Local	0	0	1
IN	Local	0	0	0
IN1	Local	0	0	0

3.3 Deploy and execute the MQ Application

To complete the full processing of the scenario, use the ProcessImages application that is provided.

1. Deploy the ProcessImages application.

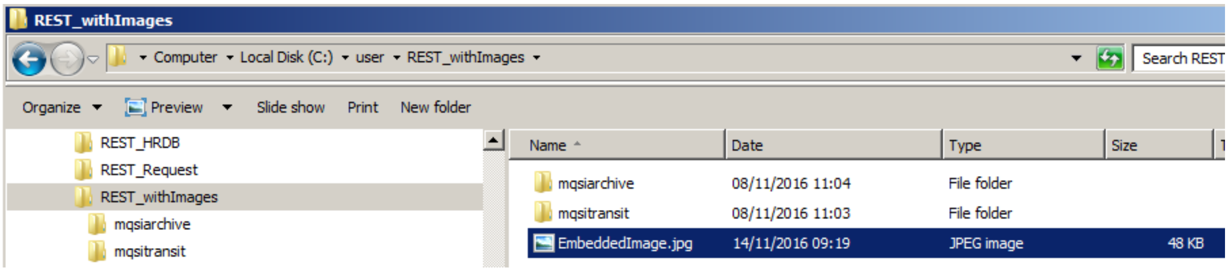


2. From the tests throughout this lab, there should be at least 1 message waiting on the MQ queue IMAGES.EMBEDDED.OUT. These will be processed immediately when this application is deployed.

   The messages, containing the original image attachment, will be written to file.

   In Windows Explorer, navigate to c:\user\REST_withImages. You will see a newly-written file, EmbeddedImage.jpg, which contains a copy of the original data that was attached to the REST request.

   Also, note that the mqsiarchive folder contains several older copies of the same file.These have filename components based on the current date and time, and have been created as a result of the properties specified on the FileOutput node in the WriteImagesToFile message flow.

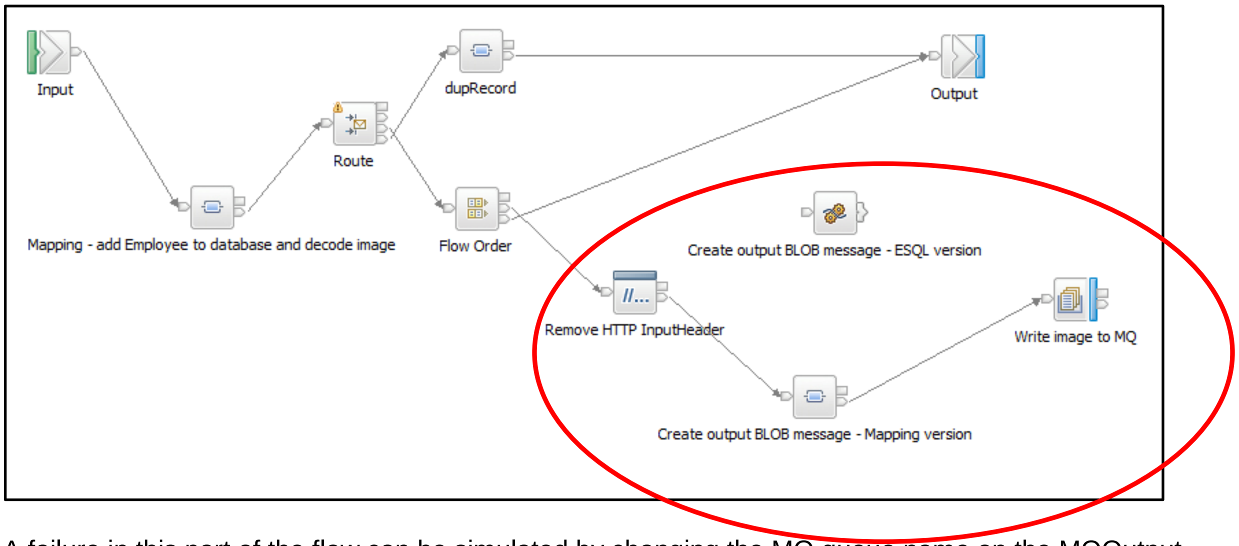


Name	Date	Type	Size
mqsiarchive	08/11/2016 11:04	File folder	
mqsitransit	08/11/2016 11:03	File folder	
EmbeddedImage.jpg	14/11/2016 09:19	JPEG image	48 KB

3.4 Error Processing

This section of the lab illustrates one aspect of failure processing.

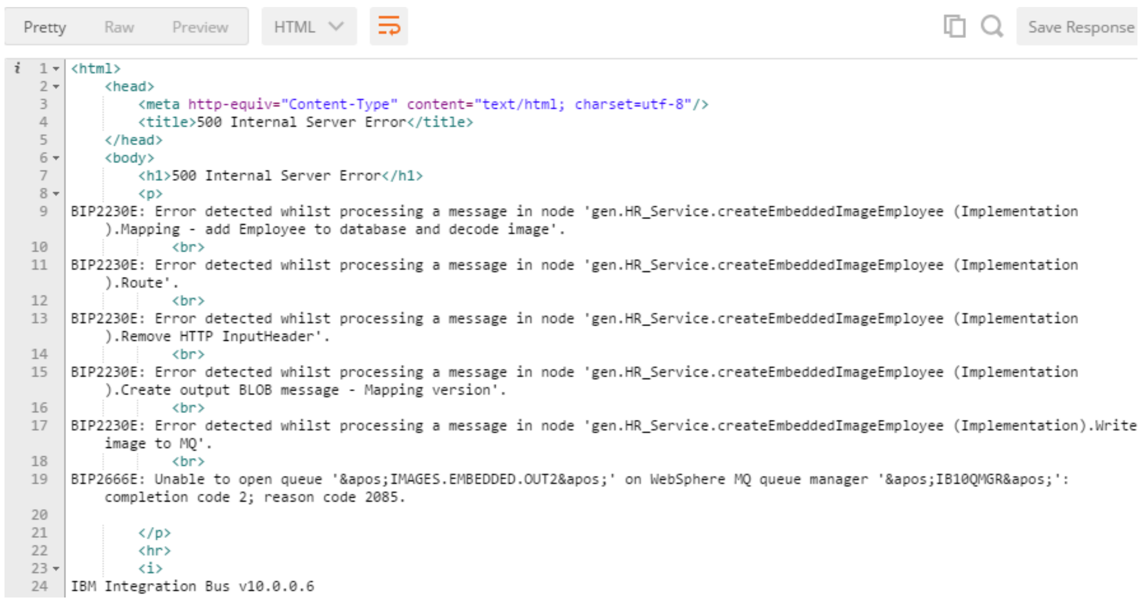
If the part of the flow connected to the FlowOrder First output terminal fails (highlighted below), then a failure message should be returned to the client.


Input
dupRecord
Output
Route
Mapping - add Employee to database and decode image
Flow Order
Create output BLOB message - ESQL version
Remove HTTP InputHeader
Write image to MQ
Create output BLOB message - Mapping version

2. A failure in this part of the flow can be simulated by changing the MQ queue name on the MQOutput node. Change the queuename to a queue that does not exist, and redeploy the flow.

3. Send a new message with Postman, as before.

   The flow will terminate, and the Second terminal of the FlowOrder node will not be invokes. The abnormal termination information will be returned to the client.



```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title>500 Internal Server Error</title>
    </head>
    <body>
        <h1>500 Internal Server Error</h1>
        <p>
BIP2230E: Error detected whilst processing a message in node 'gen.HR_Service.createEmbeddedImageEmployee (Implementation
    ).Mapping - add Employee to database and decode image'.
            <br>
BIP2230E: Error detected whilst processing a message in node 'gen.HR_Service.createEmbeddedImageEmployee (Implementation
    ).Route'.
            <br>
BIP2230E: Error detected whilst processing a message in node 'gen.HR_Service.createEmbeddedImageEmployee (Implementation
    ).Remove HTTP InputHeader'.
            <br>
BIP2230E: Error detected whilst processing a message in node 'gen.HR_Service.createEmbeddedImageEmployee (Implementation
    ).Create output BLOB message - Mapping version'.
            <br>
BIP2230E: Error detected whilst processing a message in node 'gen.HR_Service.createEmbeddedImageEmployee (Implementation).Write
    image to MQ'.
            <br>
BIP2666E: Unable to open queue '&apos;IMAGES.EMBEDDED.OUT2&apos;' on WebSphere MQ queue manager '&apos;IB10QMGR&apos;':
    completion code 2; reason code 2085.

        </p>
        <hr>
        <i>
IBM Integration Bus v10.0.0.6
```

END OF LAB GUIDE