Enterprise Application Integration

If you have disparate technologies like Java, DB, SAP etc in order to integrate in a single secure manner you need EAI

ESB is how to implement the concept of EAI.
Advantages are message transformation, message routing, message enrichment, protocol conversion.
CastIron (Cloud ESB) + IIB = ACE/ACP (ACP is on the Cloud). Upto V9 WMB you could run older versions of WMB code and WESB. IBM Integration Bus represents the IBM strategic Enterprise Service Bus offering and is the successor product for existing clients of both IBM WebSphere Message Broker and IBM WebSphere Enterprise Service Bus. From V10 IIB does not need MQ but now can install MQ on a separate server and connect remotely. IIB (V10) has Salesforce and NoSQL database connectivity.

Message Transformation

Application A produces XML format, but Application B accepts JSON

A -XML-> ESB -JSON-> B

Formats include XML, JSON, SOAP, CSV, TAG delimited

Transformation using XSL in a message flow

Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet, and to set the Message domain, Message model, Message, and Physical format for the generated message.

XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document. As of August 2022, the most recent stable version of the language is XSLT 3.0. The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

The xmlns:xsl="http://www.w3.org/1999/XSL/Transform" points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute version="1.0".

Message Routing

Application A -XML-> ESB -> B (XML)

Application A -JSON-> ESB -> C (JSON)

Suppose you have a Route node preceding the RCD node. Within it you can set XPaths. These are used to extract the header information. In the Postman request you can set the content type to XML or JSON. Then the XPath dictates which node to propagate to. It might send the message to the RCD where the domain is set.

Filter node

Use the Filter node with an ESQL statement to determine the next node to which the message is sent by this node. Do not use the ESQL code that you develop for use in a Filter node in any other type of node.

RouteToLabel node

Use the RouteToLabel node after a Compute node or a JavaCompute node for complex routing. Define a list of destinations in a Compute or JavaCompute node that are acted on by the RouteToLabel node. The RouteToLabel node interrogates the destinations and passes the message on to the corresponding Label node.

Message Enrichment

Application A -Hi-> ESB (adds ‘how are you?’) -> B (Hi how are you?)

Protocol conversion (MQ, File, http, SOAP, SMTP, POP3, JMS, SFTP)

Application A -HTTP-> ESB (HTTP to FTP) -> B (FTP)

SOA (Service Oriented Architecture)

Reusable service, loosely coupled, independent SOAP API based on XML. REST API based on JSON

Toolkit

IIB only needs MQ if you are using MQ nodes. Before IIB v9 it was called WMB where MQ is a prerequisite. The QMGR contains Integration Node configuration. IIB v10 MQ is not a prerequisite. Right click on the Application -> New message flow name LargeMessages

Application vs Integration Project

Both are a Container, but the Application provide isolation since you can use the resources it contains. This is needed for Microservices where the MS is contained.

ESQL native to IIB

Before you create your ESQL message flow you package it into a schema. An example schema is com.test.tran. This concept is akin to JAVA packages. An Integration Project's resources are reusable, but an Application's resources are contained and hence isolated.

CREATE COMPUTE MODULE XMLToJsonMsgFlow_Compute

"CREATE MODULE" creates a module, which is a named container associated with a node. Modules, functions & procedures contained by a schema must all have unique names.
Modules for the Compute node, database node & filter node must all contain exactly 1 function called Main. This function should return a Boolean. Main is the entry point used by a message flow node when processing a message.

```
CREATE COMPUTE MODULE FileNode1_Compute
	CREATE FUNCTION Main() RETURNS BOOLEAN
	BEGIN
		-- CALL CopyMessageHeaders(); // This will only copy the Properties & MQMD. NOT the XMLNSC as I < J.
		-- CALL CopyEntireMessage();
		RETURN TRUE;
	END;

	CREATE PROCEDURE CopyMessageHeaders() BEGIN
		DECLARE I INTEGER 1;
		DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
```

```
            WHILE I < J DO
                  SET OutputRoot.*[I] = InputRoot.*[I];
                  SET I = I + 1;
            END WHILE;
      END;

END MODULE;
```

The **COALESCE** function evaluates its parameters in order and returns the first one that is not NULL

CARDINALITY returns the number of elements in a list; here the correlation InputRoot is pointing to the message tree which contains the Properties, MQMD and XMLNSC so CARDINALITY is 3. InputRoot is a field reference with a [] array indicator.

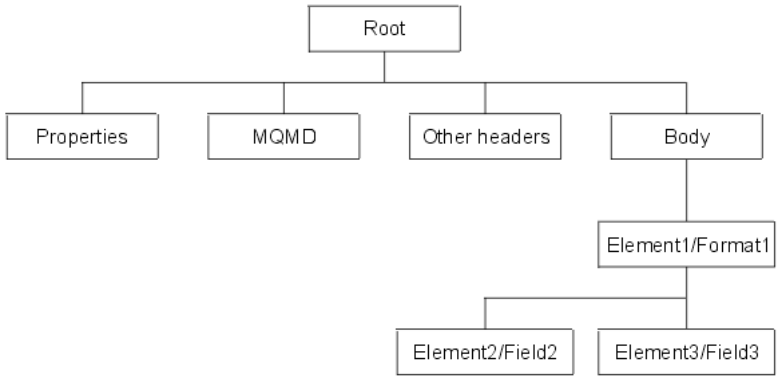
InputRoot.*[] allows you to refer to the array of all children of the root element using a path element of *

InputRoot.*[<] last child of root of the input message. As you can see from the diagram it is Body. The last child element beneath the root of the message tree is always the message body. For example, XMLNSC or JSON.

InputRoot.*[1] first child of the root of the input message i.e. the message properties.

Converting XMLNSC to JSON

If your ESQL has




```
SET OutputRoot.JSON.Data = InputRoot.XMLNSC;
```

This will only work if you are not sending multiple arrays of elements in your XML. The same applies to JSON to XML. Your output root will not have well formatted message trees. That means the nodes that are downstream will fail while formatting the data 04JAN2021-IIBFileNodeBARMQReplyTryCatchFlowOrder

Difference between DECLARE varName CHAR FIELDNAME() & DECLARE varName REFERENCE TO

If your JSON looks like

{“root”:{

“Number1”:4,

“Number2”:2,

“MathOp”:”+”,

}

}

```
DECLARE InRef REFERENCE TO InputRoot.JSON.Data.root;
```

The InRef is a reference variable pointing to part of the message tree (root).

```
DECLARE mathOp CHAR InRef.MathOp;
```

mathOp is +

```
DECLARE Operation CHAR FIELDNAME(InRef.*[<]);
```

The FIELDNAME returns the name of the field identified by source_field_reference inside () as a character value. If the parameter identifies a nonexistent field, NULL is returned. Operation is now MathOp (a string of type CHAR).

The MOVE statement

```
 MOVE InMgrref NEXTSIBLING REPEAT TYPE NAME;
```

This moves InMgrref in the direction of the next sibling relative to the current position if there is a sibling. If not LASTMOVE returns false. The TYPE and NAME clauses mean target is moved to a field with the given type, name. Fields that do not match the criteria are skipped over.

LEAVE statement

The LEAVE statement stops the current iteration of the containing WHILE, REPEAT, LOOP, or BEGIN statement identified by Label. The containing statement's evaluation of its loop condition (if any) is bypassed and looping stops.

Syntax

LEAVE *Label*

Examples

In the following example, the loop iterates four times:

```
DECLARE i INTEGER;
SET i = 1;
X: REPEAT
     IF i>= 4 THEN LEAVE X;
END IF;
SET i = i + 1;
UNTIL
     FALSE
END REPEAT;
```

A field reference consists of a correlation name, followed by 0 or more Path Elements separated by periods (.). The correlation name identifies a well-known starting point and must be the name of a constant, a declared variable, or one of the predefined start points; for example, InputRoot. The InputRoot is the root of the input message. The OutputRoot is the root of the output message. Message trees could refer to the input message or output message. The tree structure created by the parsers is independent of any message format (e.g. XML). The exception to this is the subtree created as part of the message tree to represent the message body. For example, XMLNSC, JSON or DFDL.

Message Tree structure is populated with the contents of the input message bit stream. It has a root element InputRoot. Each tree is made up of elements. Root has a number of child elements.

The Compute node has an input message assembly and at least 1 output message assembly. Configure the Compute node to determine which trees are included in the output message assembly. If you want the output message assembly to contain a complete copy of the input message tree, you can code a single ESQL SET statement to make the copy.

```
CREATE PROCEDURE CopyEntireMessage() BEGIN
      SET OutputRoot = InputRoot;
END;
```

This procedure copies the entire contents of the input message tree (Properties and MQMD which are the header & XMLNSC which is the payload) to the output message.

If you want the output message to contain a subset of the input message tree, code ESQL to copy those parts only.

```
SET OutputRoot.XMLNS.ResAdd.sp1:addC=InputRoot.XMLNSC.sp1:ReqAdd.sp1:intA + InputRoot.XMLNSC.sp1:ReqAdd.sp1:intB;
```

The message tree is always present and is passed from node to node in a single instance of a message flow. The message tree includes all the headers, in addition to the message body.

The header of the message tree is the Properties & MQMD. The payload is the XMLNSC in the diagram and is also called the Body tree which is a structure of child elements that represents the message content (data) and reflects the logical structure of that content. The body tree is created by a body parser. Each element in the parsed tree is either a name, value or name-value element.

Inside the main function after the 2nd call:

```
CREATE LASTCHILD OF OutputRoot DOMAIN('JSON');
 CREATE FIELD OutputRoot.JSON.Data;
```

LASTCHILD is a type of FIELD clause. "CREATE LASTCHILD OF" target navigates to the target field and adds a new field as it's rightmost child, displacing the previous last child to the left. The DOMAIN('JSON') associates the new field with a new parser of the specified type i.e. JSON.

```
 DECLARE outref REFERENCE TO OutputRoot.JSON.Data;
 CREATE FIELD outref.Purchases IDENTITY (JSON.Array)Purchases;
SET outref.Purchases.Item[1].Description = inRef.Description;

      -- This creates JSON message under the Data element owned by JSON parser root. This creates a child element of the OutputRoot
      -- as a JSON. Since the JSON tree structure is JSON and data this assigns InputRoot.XMLNSC to OutputRoot.JSON.Data.
```

You can manipulate messages that belong to the JSON domain, which are parsed by the JSON parser. The code transforms the OutputRoot message to JSON. The OutputRoot... statement produces a message tree.

If you include a FIELD clause the field specified by TARGET is navigated to.

SET assigns a value to a variable.

XMLNSC parser is guided by the XML Schema (describes the shape of the message tree which is a logical model). XMLNSC is the preferred domain for parsing all XML because of it's high performance, reduced memory used by the logical message tree created from the parsed message; which has discarded non-significant whitespace, mixed content, comments, processing instructions & embedded DTDs. XMLNSC parser can operate as a model-driven parser and can validate XML messages against XML schemas, to ensure XML messages are correct.

Local, UDP (AKA External) & shared variables

Local variable scope is within the Compute node. UDP scope is throughout the message flow. Shared scope is through the execution group AKA the Integration Server

even if the message flow completes. When the next transaction starts the shared variable is available.

MQInputNode mode. How your messages will be processed

Under Properties then Advanced there is a Transaction mode which relates to how your messages are going to be processed.

This could be Yes (message is received but if there is an exception the original message will go to a backout queue; which if defined is at the queue level or DLQ; defined at QMGR level else it will loop), No or Automatic. Imagine there is a problem in the compute node or suppose the queue on the MQ output has a problem then the message will go to the backout queue defined at queue level. If not defined, then it goes to DLQ defined at QMGR level. If no DLQ then continuous loop. An exception could occur if the wrong data source is defined at the Compute Node in between the MQInputNode and MQOutputNode. The MQInputNode has a Basic tab under properties which has a Queue Name field. If say IN.LQ is the value of that field then you need to define the backout queue for that queue using MQ Explorer.

Yes: If you select Yes, the incoming message is received under sync point.

The syncpoint is defined as below:

The decision to commit or back out the changes is taken, in the simplest case, at the end of a task. However, it can be more useful for an application to synchronize data changes at other logical points within a task. These logical points are called sync points (or synchronization points) and the period of processing a set of updates between two sync points is called a unit of work. Therefore, a unit of work is named as processing between two sync points. From my understanding, that unit of work is a transaction actually.

```
----------syncpoint-------
          |             BEGIN
          |
          |
      a unit of work
          |
          |
          |             COMMIT
----------syncpoint-------
          |             BEGIN
          |
          |
      a unit of work
          |
          |
          |             COMMIT
----------syncpoint-------
```

No: If there is a problem in the compute node or suppose the queue on the MQ output has a problem then the message won't go anywhere. It will be lost.

MQ Output

29DEC2020-IIB Timeout_LabelRouting_MultiQueues. You can send the message to multiple queues in ESQL. You need to change the Compute mode to LocalEnvironment and Message in the Compute node properties as well as the MQ Output's Destination mode to Destination List. Use the command:

```
SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName='OUT';
SET OutputLocalEnvironment.Destination.MQ.DestinationData[2].queueName='OUT1';
```

MQ Reply

04JAN2021-IIBFileNodeBARMQReplyTryCatchFlowOrder 24 minutes. The client system sets the name of the queue the flow has to reply to. RFHUTIL allows you to set the ReplyTo header setting.

IIB Node Connecting to Database using ODBC pt1

Use the command console to associate a DSN with a broker instance. You create an ODBC connection to the data source DBNAME with the command:

mqsisetdbparms IB9BROKER -n DBNAME -u -p

You now need to reload the configuration since the new association of DBNAME to IB9BROKER:

mqsireload IB9BROKER

Then check the connection using "mqsicvp IB9BROKER -n DBNAME"

IIB Node Connecting to Database using ODBC pt2

IIBGURU entered:

$ db2start
$ db2
db2 => connect to sample
db2 => SELECT * FROM EMPLOYEE

Create a new Application name DBINTER_APP. Then right click on DBINTER_APP then create a message flow DBINTER_MF. From the palette add MQInput and MQOutput nodes. In between add some ESQL in a Compute node. MQInput out connects to the in of the Compute node. Compute out connects to the in of the MQOutput node. MQInput node will use the XMLNSC parser to read XML records

```
9001
TRUMP
New York
```

…that will be INSERTED into DBNAME. The Compute node has Data Source DBNAME. Double click on the node to bring up the ESQL editor.

Uncomment — CALL CopyMessageHeaders(); so that we can send a success message to the out queue.
Compare

```
INSERT INTO EMPLOYEE VALUES(101,'IIBGURU','CALIFORNIA')
```

With what we use in ESQL Compute Node.

```
        INSERT INTO Database.iibguru.EMPLOYEE VALUES(InputRoot.XMLNSC.EMPLOYEE.ENO, InputRoot.XMLNSC.EMPLOYEE.ENAME,
InputRoot.XMLNSC.EMPLOYEE.ECITY);
        — You can use CTRL space to have content assist help with the syntax
```

The above could be written as:

```
        DECLARE InRef REFERENCE TO InputRoot.XMLNSC.EMPLOYEE;

        INSERT INTO Database.iibguru.EMPLOYEE VALUES(InRef.ENO, InRef.ENAME, InRef.ECITY);
```

We are getting the values from the Input message tree where the domain is XMLNSC, EMPLOYEE is the root child. “Database” is a keyword & the iibguru is the schema in which EMPLOYEE table exists. Here the schema is the user name. It will use the user name as the schema if you have not created the schema before creating a table in the database.

When you save the ESQL in the Compute node you get a warning “Unresolvable database table reference ‘Database.iibguru.EMPLOYEE’”. This is because it only resolves and finds EMPLOYEE during execution. The successive command after the INSERT will only run if the INSERT executes correctly. It is:

```
        SET OutputRoot.XMLNSC.DBINSERT.STATUS=‘SUCCESS’;
```

Introduce a breakpoint after the MQInput. Since you have the breakpoint you have to launch the debugger (right click on execution group) then add a message from the XML file using RFHUTIL. In the Data tab of the RFHUTIL you can parse the file to see if it is syntactically correct under Data Format. Once you write the XML file to the IN queue the Debug perspective is automatically launched. You will see multiple threads. Thread suspended at DBINTER_MF at connection. You can click on the ‘Edit Source Lookup Path’ button. From there you add a container to the source lookup path. Since we are running a MF we choose the Message Flow Container, choose our DBINTER_APP application which contains our message flow. Under Variables you can see the Message. As you know the Message has Properties then MQMD then Other headers then the Body which has

```
        XMLNSC    |
                |
                V
                EMPLOYEE|
                          |
                         V
                        ENO (9001), ENAME (TRUMP), ECITY(New York)
```

We can go inside the scope of the Compute node by clicking on the thread in the Debug perspective, then click on Step into code icon which will open up the ESQL of the next Compute node. You will reach the CALL CopyMessageHeaders(); You can then click on the Step over icon which will run the CALL then take you to the INSERT. The CALL will copy the headers to the output. EVEN though the INSERT has been executed the change has not been committed to the database. This is because the

```
INSERT INTO….
SET OutputRoot…
RETURN TRUE;
```

Are considered as 1 unit of work. Only until every command in the ESQL happens without error & the whole flow completes will the INSERT be committed. Else a rollback will occur. This is controlled by the Transaction property of the Compute node. The possible values are Automatic (default) or Commit. The latter will commit the INSERT once the Compute node completes successfully. It will not wait for the whole flow to complete.

PATH AREA.CIRCLE # this allows you to reference ESQL in another broker schema

Oracle DB ESQL

DSN TEST_DSN, Schema is SYSTEM

```
DECLARE EMP ROW;
        — ROW is a datatype that holds a tree structure
        SET EMP.Result[] = SELECT * FROM Database.SYSTEM.EMP_DETAILS AS R WHERE R.EMPID=InRef.id;
```

We then send this XML:

```
   SET OutputRoot.JSON.Data.Employee.ID = EMP.Result.EMPID; //only works with a single value not multiple
```

PASSTHRU with EVAL

https://youtu.be/0CX9--Y2jOk?si=0Mx5cJC8IDdW2GL1

Example of PASSTHRU where there are 2 procedures with the same name but in ESQL files contained in different schemas. How can we invoke them dynamically at runtime? We can use EVAL to run the procedure that is connected to a key value in some incoming XML or JSON. For example if the value is Professional then invoke the EmpDetails procedure in the Pro schema. If it is Personal invoke the EmpDetails in the Personal schema. https://youtu.be/0CX9--Y2jOk:

```
SET Emp.EmpRecord[] = PASSTHRU(                                        ||’’’’||EmpID||’’’’);
SET Emp.EmpRecord[] = PASSTHRU('SELECT * FROM EMPLOYEE WHERE EMPNO='||""||EmpID||"");

DECLARE DbRecords REFERENCE TO Emp.EmpRecord;

CREATE FIELD OutputRoot.JSON.Data.Employee.{EmpInFoType};
DECLARE outDetailsRef REFERENCE TO OutputRoot.JSON.Data.Employee.{EmpInFoType};

IF CONTAINS(EmpInFoType, 'Professional') THEN
SET SchemaName = 'EmpProDet';
ELSEIF CONTAINS(EmpInFoType, 'Personal') THEN
SET SchemaName = 'EmpPersonDet';
END IF;
```

EVAL('CALL '||SchemaName||'.EmpDetails(outDetailsRef, DbRecords);');

RETURN TRUE;

This video also shows you how to use EVAL. (https://youtu.be/0CX9--Y2jOk?si=h0r2PTL2h-y_qw-x). EVAL saves you from having to write lots of if statements by evaluating the expression on the fly using variables.

Example:

```
SET OutputRoot.JSON.Data.Result = EVAL('num1'||mathop||'num2');
```

Instead of:

IF mathop = '+' THEN
SET `OutputRoot.JSON.Data.Result = num1 + num2;`

Shared variable and ATOMIC block

https://youtu.be/u5-r4H6PvIY?si=kVKM9JrdJnXVlj6p

```
DECLARE MYROW ROW; --A row variable contains an array
```

ROW SHARED variables can hold complex tree structure. A shared ROW can contain the result of a SELECT on the database table being cached.

```
DECLARE CACHE SHARED ROW; --For example, a database table called "AIRPORTS" contains two columns, "CODE" and "CITY". This code loads the
       cache:
SET CACHE.AIRPORT[] = SELECT A.CODE, A.CITY FROM Database.AIRPORTS AS A;
```

The CACHE variable will be populated like this:
-- CACHE.AIRPORT[1].CODE = AAA
-- CACHE.AIRPORT[1].CITY = Anaa
-- CACHE.AIRPORT[2].CODE = AAB
-- CACHE.AIRPORT[2].CITY = Arrabury

Accessing ROW SHARED variables is much faster than retrieving the data from the Database directly depending the amount of data retrieved. The first time after the ESQL code is run CACHE shared variable is in memory. The next time it is invoked it does not have to reload from the database as it is cached.

The problem with this cache structure is that it doesn't scale. A user trace will show that SELECT scans the table sequentially until it finds a row that satisfies the WHERE clause. As the table grows, the search gets slower. There comes a point when it's faster to drop the cache and go to the database each time. Ref: https://www.websphereusergroup.co.uk/wug/presentations/38/EfficientCaching_-V2.pptx.pdf

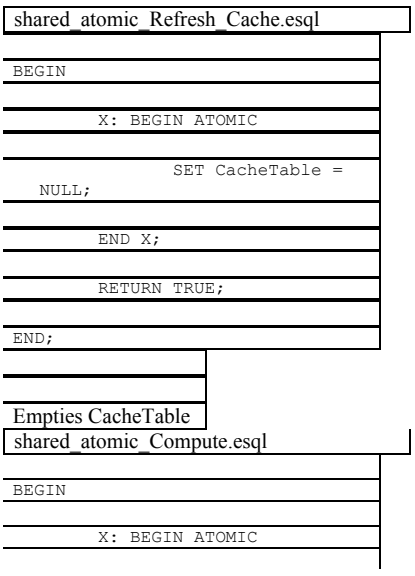
Whatever is stored in a shared variable is held in cache memory until we refresh the execution group, application or application flow using 'mqsireload' for example. Other message flows in the same execution group access the shared row variable value set in a message flow.

Once a cached shared row is populated then every time the flow will run it will only have what was loaded at the time of the SQL query. This means that the underlying database values may change but this won't be reflected.
In a PROD environment you can't run 'mqsireload' all the time as that would impact LIVE services. One option is to create a CACHE queue on the queue manager. This could be referenced in the properties of the same flow but in a separate MQ Input node connected to a Compute node. Within the Compute node you can empty the shared row variable:

```
SET CACHE = NULL;
```

When ESQL code in other compute nodes access the variable, they will see it is empty, which prompts another SELECT against the database. If the "SET CacheTable = NULL;" is executed just before "SET OutputRoot.XMLNSC.Data = CacheTable;" then the output message assembly will have no data.

You have to use atomic blocks (https://youtu.be/u5-r4H6PvIY?si=V3_4tA09WjTF0WRn) within the same schema to ensure that threads are executed serially (only 1 thread is executed at a time). That way we can update the shared row variable in a thread safe manner.



```
                IF NOT
   EXISTS(CacheTable.[]) THEN

                        SET
   CacheTable.Result[] =
   PASSTHRU('SELECT
   R.FIRSTNME,R.LASTNAME FROM
   EMPLOYEE AS R');

                        SET
   OutputRoot.XMLNSC.Data =
   CacheTable;

               ELSE

                        SET
   OutputRoot.XMLNSC.Data =
   CacheTable;

               END IF;

        END X;

        RETURN TRUE;

END;
```

Both atomic blocks must use the same X label

One advantage of the Global Cache over ESQL shared variables is that the cache can be shared between message flows, integration servers / execution groups, and integration buses / brokers (recall that the scope of ESQL shared variables is the message flow). However, because Global Cache uses the JVM to store data (Udemy 18) it is slower than shared variables which use cache.

The THE function returns the first element of a list.

Syntax
THE(ListExpression)
If *ListExpression* contains one or more elements; THE returns the first element of the list. In all other cases, it returns an empty list.

Restrictions
ListExpression must be a SELECT expression.

DATE TIME TRANSFORMATION

IBM help "formatting and parsing date times as strings" under help in the toolkit (Search for parsing date and time). How do we interpret date formats? 12-09-2015 12:05pm. This is called a timestamp as it has a time component The first 2 digits are the day in the month. We want to transform to :
Month in 3 letters/day in the month with 0 as padding/4 digits year:hours,mins,seconds

```
        DECLARE INDATE CHARACTER '12-09-2015 12:05pm';

        DECLARE INDATEFORMAT CHARACTER 'd-MM-yyyy h:mma';

        --This tells us how to interpret the string in a format IIB understands

        DECLARE STDDATE TIMESTAMP CAST(INDATE AS TIMESTAMP FORMAT INDATEFORMAT);

        --We have to change it to a standard date using cast

        DECLARE OUTDATEFORMAT CHARACTER 'MMM/dd/yyyy:hh,mm,ss';

        DECLARE OUTDATE CHARACTER CAST(STDDATE AS CHARACTER FORMAT OUTDATEFORMAT);
```

He then deploys the application to the execution group. Iibguru then sends a blank message to the IN queue that mqinput node that connects to the esql above to trigger the code. The thread stops at the breakpoint in between. Then he steps into the code, then steps over. He can see the values of the DECLARED variables as they change

INTERVAL DATATYPE

How many either days, months, years (up to you) between either 2 timestamps or dates. Uncomment — CALL CopyMessageHeaders(); so that we can send a success message to the out queue.

```
        DECLARE INDATE DATE '2009-09-17';
        — The type is DATE not CHARACTER since it is a standard date format. When you run with breakpoints you will observe the variable
as INDATE:DATE:java.util.GregorianCalendar[time=12531….] & OUTDATE as 2009-10-17

        DECLARE OUTDATE DATE INDATE+INTERVAL '1' MONTH;
```

If you want to see what day this date is:

```
DECLARE DAYFORMAT CHARACTER 'EEE';

DECLARE DAYNAME CHARACTER CAST(OUTDATE AS CHARACTER FORMAT DAYFORMAT);
```

—DAYFORMAT is EEE, DAYNAME is Sat
—if DAYFORMAT is EEEE, DAYNAME is Saturday
—if DAYFORMAT is 'EEEE W', DAYNAME is Saturday 3 where 3 is the 3rd Saturday of that month

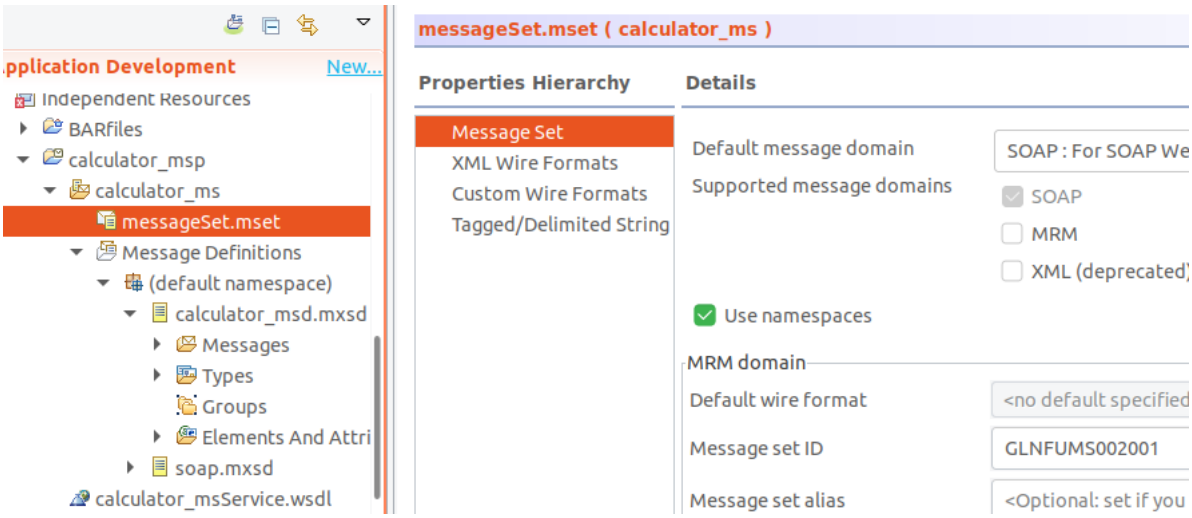
```
DECLARE SUBDATE CHARACTER (CURRENT_DATE-INDATE) YEAR;

—This will return an interval string not a DATE type. Hence we use CHARACTER not DATE.
SUBDATE Value is INTERVAL '8' YEAR

DECLARE SUBDATE CHARACTER (CURRENT_DATE-INDATE) YEAR TO MONTH;
— SUBDATE Value is INTERVAL '7-09' YEAR TO MONTH



If you want the differences in hours then change the INDATE declaration from DATE type to TIMESTAMP type:
DECLARE INDATE TIMESTAMP '2009-09-17 13:53:25';

You also need to change the SUBDATE to:
DECLARE SUBDATE CHARACTER (CURRENT_TIMESTAMP-INDATE) HOUR;
```

Transformation Extender

CloudPak is a pack made up of IIB, MQ, WTX, DataPower & API Connect. WTX is coding free transformation but in IIB we can do transformation in 5 coding languages.
Data transformation engine at the core of IBM TX. Consider the following data stream for which ITX has an adapter Paul Brett Operations 1/4/1970
The data stream is read into a type tree. Consider this tree to be a type of bucket. This bucket can hold further buckets. This can be transformed into: Paul Brett Operations 1970-04-01
The output adapter sends the XML to either MQ, a file or something else. WTX protocol support is less.

Message Set, Message definition creation

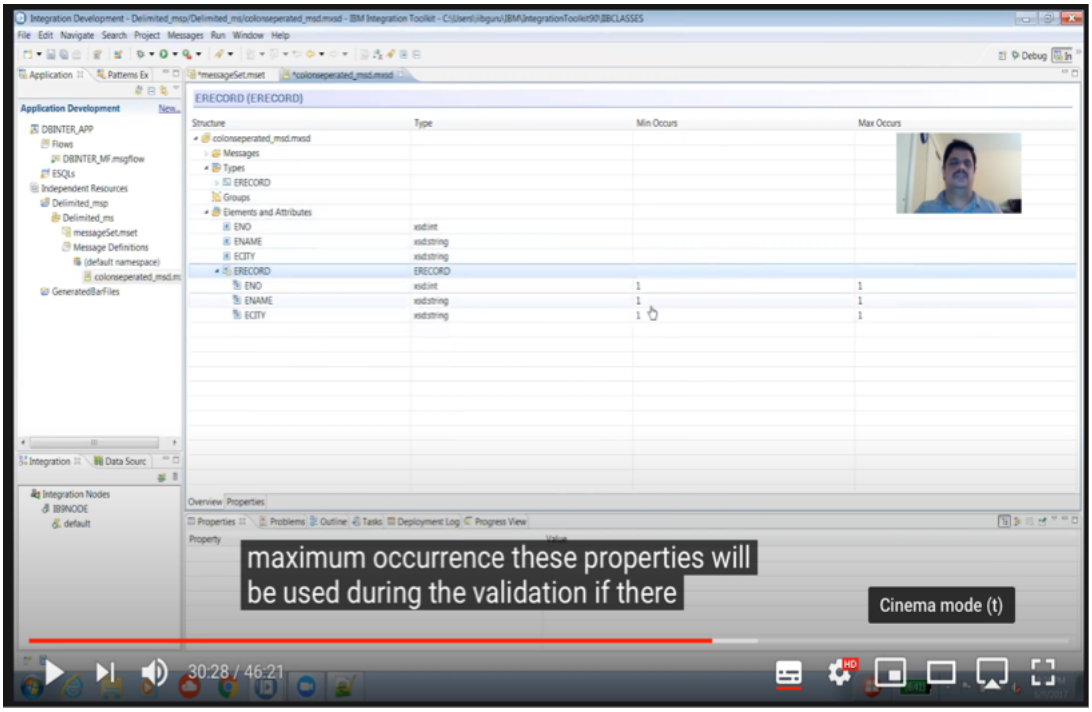
https://youtu.be/ZC_uDGzhKIs?si=Bz4sfjpp9vF-Wpbs. Udemy Section 15:60. For XML and JSON Message Modelling is optional. But for CSV, Flatfile, EDI, HL7, SWIFT, COBOL etc it is mandatory. Message model schema files are the preferred way to model messages for most data formats. How are you going to create these and implement them as part of the message flow? A request or response message needs to be defined. What are the individual fields or field values contained therein? Parsing is segregating the different parts of the message and interpreting what each part means.
A message set can contain 1 or more message definitions. A message set has a unique ID.



The message definition defines the field names, data types, delimiters. This definition helps with say a message transformation. Consider the following data
123:raju:detroitCRLF
The carriage return, line feed can be seen in Notepad Plus Plus. CRLF appears in Windows, LF appears if created in Linux.
In IIB Toolkit go to New -> Other -> Message Set. Give it a name of Delimited_ms, give Delimited_msp for the project name. Click next then select the type of message data most appropriate. We choose Text data. Take a note of the next windows message domains (MRM which is the default), wire formats (Text1), and Schemas (None). You will see your newly created messageSet. Change the default wire format to Text1.

Remember message set ID is different from MQ message ID. The latter is an MQMD (message queue message definition) property.

You then right click on Message definition -> New -> Other -> Message definition -> Message Definition File
Click Next and name the file colonseparated_msd. Right click on Elements and Attributes -> create Global element (lowest level)
The 3 fields combine to make a single record. That record should be given a name and the name added to Types. Add it as a Complex type. In order to tell IIB that ERECORD is a composite of the 3 fields right click on ERECORD then Add Element Reference.



You need to go to Properties of ERECORD then select all elements delimited then add colon for delimiter. When you test this using MQInputNode you need to set the correct wire mode else you will get "Invalid Wire format retrieved". This message is a sign that you have not given Broker the correct definitions for it to interpret the text received.
We need the message definition we created to be referenced correctly in the MQInputNode -> Input Message Parsing. Under Message model it should show you the message set but it does not unless you let the application have access to the MS which is an independent resource. You need to change the applications properties -> Project References -> Choose Delimited_msp. NOW you can select Delimited_ms in the Input Message Parsing. This also allows you to select ERECORD as the Complex message definition. Physical format is the Text1 wire format.
When you see the Variables under the running thread you will not see the ERECORD Complex type since it has been replaced by the MRM. Any top-level Complex type of a text delimited message definition is going to become MRM. You will see the children of MRM (ENO, ENAME,ECITY).

Message Set, Message definition creation for XML

Message set will only tell you the format. But the actual schema XSD will be in the Message definition. In the New Message Set wizard after giving the message Set and project names click Next. Then choose "XML documents" as the type of message data then Next. Wire formats is None and Schemas are None. As ever the next step is to create some message definitions in the same message sets. You can create them from XSD files.

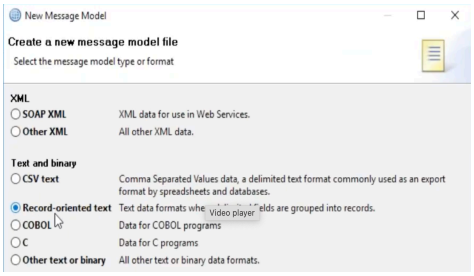
Message modelling

JAN192021-IIB JCN_ForLoop&JDBC&ModellingPt1 18 minutes. The modelling of a message can be achieved in the message set and definition as above or within an IIB Message Model as introduced in version 8. You can reference the Message Set and one of it's Definition objects in code. The Message Model is the more advanced concept over Message Set and Definition because you can test your model including performance without deployment ('Test Parse Model' in the model xsd window JAN212021-IIB ModellingPt2&Mapping). For example, you can model a CSV file. The model should be in a container like a library or an application.

DFDL is a more advanced version of MRM. If you model the data using the Message Set and message definition, then the parser is MRM. However, if you model your data using the Message Model then the parser is DFDL. Refer to 14DEC2020-IIBMSetMDef.mp4 27:24 and IBM/IIBT10/workspace/MindMajix/David/CSV_XML_MessSetDEMO_MF_Compute.esql
In your MQInput node there is an Input Message Parsing tab. When you are expecting MRM, you need a Message value of say {}:IN. This is in your Message definition. You will see this appear as MessageType in your Message -> Properties in debug. If you want to convert from, say XML, to CSV you cannot set the MQInput node properties in the MQOuput node. You need a Compute node with this:

```
SET OutputRoot.Properties.MessageSet='csvSingleLine';
SET OutputRoot.Properties.MessageType='{}:TargetMessag';
SET OutputRoot.Properties.MessageFormat='Text_CSV';
```

If your incoming data contains a header and a trailer, then use record-oriented text and select 'The first record is a header' in the wizard.



JAN212021-IIB ModellingPt2&Mapping 12 minutes. We saw the transformations with ESQL and the JCN. If you don't want to write a single line of code, then the mapping transformation node helps. However, you cannot do complex transformation using the mapping node. You can only do minimal types of transformations like converting XML to JSON, fields mapping, string values or static values.

You can refer to Message Models in a Mapping node. But the Mapping node cannot reference an existing message set/definition

Another fundamental point is that generally for XML & JSON 99% of the time we don't model since the description of what the data values mean is already defined (for example Edetails.xsd). But, if you are planning on doing the transformation using the mapping node, then you need to model the data in a Message Model, even if that data is XML or JSON.

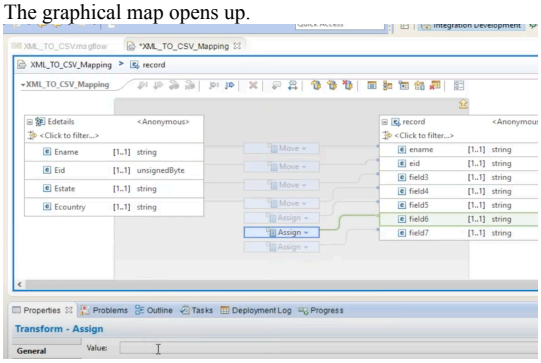
Mapping XML from an XSD

JAN212021-IIB ModellingPt2&Mapping 16 minutes. The IBM/IIBT10/workspace/LIB_MODEL/Edetails.xsd file models the data for the incoming XML. It is generated online using an existing XML file as a template. The XSD file validates the XML and also speeds up development of your integration applications by enabling ESQL content assist and graphical maps.

The MAP_PRO application references the LIB_MODEL we created. When we create a new message model file from the Edetails.xsd file we import it into our workspace library LIB_MODEL.

When we double click on the mapping node a wizard opens. We choose the 'Edetails' model as the map inputs and the 'csvone' model as the map outputs. Click next then choose DFDL as the output domain. This is correct since we created 'csvone', using DFDL.

The graphical map opens up.



The root of the left XML message assembly is Edetails. The Edetails [1..1] tells us there is at least one details group. The input XML looks like this.

```
<Edetails>
   <Ename>David</Ename>
     <Eid>478</Eid>
     <Estate>Juhu</Estate>
   <Ecountry>IN</Ecountry>
</Edetails>
```

The fields [1..1] means there must be at least one occurrence in the output assembly. But the input assembly does not move any values to those fields. That is why we Assign a dummy value in the Assign Properties -> General -> Value field.

Mapping XML with multiple records from an XSD

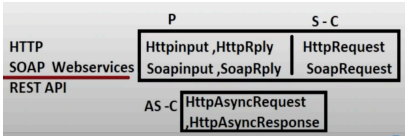
JAN192021-IIB JCN_ForLoop&JDBC&ModellingPt1.mp4. You can edit your mapping so that it has different map inputs and outputs. You can delete the existing input/output using the red cross. From this XML we modelled mulxml.xsd.

```
<Empdetail>
<Emp>
<Ename>David</Ename>
<Eid>478</Eid>
<Estate>Juhu</Estate>
<Ecountry>IN</Ecountry>
<Eage>35</Eage>
<Egend>m</Egend>
<Edoj>1222</Edoj>
</Emp>
<Emp>
<Ename>Amit</Ename>
<Eid>3269</Eid>
<Estate>Osterley</Estate>
<Ecountry>UK</Ecountry>
<Eage>52</Eage>
<Egend>m</Egend>
<Edoj>1217</Edoj>
</Emp>
</Empdetail>
```

JAN212021-IIB ModellingPt2&Mapping 35 minutes. Since both the XML and CSV model are multiple records, you go with the For Each in the Mapping GUI.

Web Services

Synchronous versus Asynchronous. https://youtu.be/N5Ky-mz6n-8



SOAP is only XML, but HTTP can handle any format be it XML, JSON, delimited, HL7 etc. SOAP is XML over HTTP or JMS HTTP + SSL = HTTPS but XML uses WS-SEC. 19DEC2020-IIBProdConsumeHTTP.mp4 23minutes

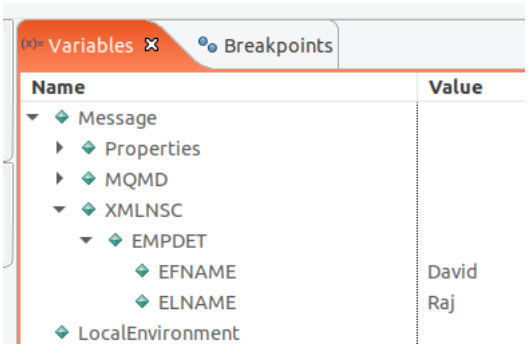
HTTPInput listens for a request whereas HTTPReply is responsible for responding to the consumer. HTTPInput listens on a port. There may be multiple web services listening on the same port. To differentiate use the path suffix for the URL

mqsireportproperties IIBGURU -e IIBGURU_EX -o HTTPConnector -r

HTTPInput can expose the service using different methods (GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS) so you can't expect just to hit with a browser.

Because the Transformation node has:

SET FNAME=InputRoot.XMLNSC.EMPDET.EFNAME



That means we are expecting the payload. GET does not have a payload. Coordination of the millions of requests going through this flow is via the LocalEnvironment -> Destination -> HTTP -> RequestIdentifier (some unique number).

In google drive '[2020-12-19] IIB ProdConsumeHTTP' the IIB Demo as a consumer and producer calling local HTTP Service and Internet Pokemon.

SOAP Gateway mode

You can operate SOAP without a WSDL in gateway mode. There are 2 operation modes "Specify WSDL interface to expose" & "Operate in gateway mode". Operation mode is the manner in which messages are routed by the SOAP Input node. If you choose gateway mode, then Validation is not enabled. In the default mode the WSDL will contain the rules to validate against.

However even in Gateway mode the SOAP Input won't accept any kind of message; it still expects SOAP format. You will get an "Exception during creation of SOAP envelope structure" if your request is not SOAP XML format. This is because the SOAP Input property's Input Message Parsing still has SOAP in the message domain setting even after choosing Gateway mode.

Exposing SOAP webservice. WSDL creation

HTML page is human consumable. But a web service is code consumable. Imagine bits of code talking to each other.
A web service is either SOAP or more recently REST API. SOAP needs a WSDL file JAN132021-IIB Containerization_SOAP_Reseq. We also need a message definition to describe the request and response data. Our calculator SOAP service will have a single message definition to define the numbers to be added and the result.

We now need to create a WSDL out of the message definition. Just right click on the mxsd then generate WSDL.
WSDL (Web Service Description Language) is like an interface. You don't care what happens internally. You only care about how to interface with the service.
At the upper level WSDL has a service, service has a binding, binding has a port type, port type has an operation, operation has a message then message has a data types.
The lowest component is data type which is akin to an XSD.
You then right click on Message definition -> New -> Other -> Message definition -> Message Definition File

You need to specify the target namespace. This allows you to separate elements with similar names. JAN132021-IIB Containerization_SOAP_Reseq 9 minutes. You also need to Generate WSDL definition. This requires a setting to allow this option in the toolkit.

REST

Client <-transfer---Representational Data---transfer>API → Server

Below ReqAdd is a resource that is represented by intA & intB and their data types.

JAN052021-IIB_REST. REST is an architectural concept. You are trying to expose HTTP operations with appropriate HTTP methods. The equivalent to the WSDL file is the swagger document. This is where you define the request and response for GET, POST etc. You also define the data types of those elements. The format is mostly JSON since it is the fastest transformation through the layers. COVID data is in JSON. In IIB the swagger is automatically generated but you can import the swagger. IIB version 10 introduced REST API wizard. When you double click the message flow the REST API editor opens. If you right click and open with the message flow editor, you will see an HTTP Input and HTTP Reply nodes either side of a route to label node. As you make changes to the REST API editor the swagger "paths" : {} gets updated with your operations as well as the message flow.

You then create a subflow for the operations in the REST API editor. Add a transformation node (Compute).

```
CAST with CCSID. InputRoot.Properties.CodedCharSetId (this will be populated by the Broker).
```

```
SET OutputRoot.JSON.Data.Result = CAST(InputRoot.JSON.Data.One AS INTEGER CCSID InputRoot.Properties.CodedCharSetId ENCODING
     InputRoot.Properties.Encoding) + CAST(InputRoot.JSON.Data.Two AS INTEGER CCSID InputRoot.Properties.CodedCharSetId ENCODING
     InputRoot.Properties.Encoding);
```

```
SET OutputRoot.JSON.Data.Result = CAST(InputRoot.HTTPInputHeader.One AS INTEGER)*CAST(InputRoot.HTTPInputHeader.Two AS INTEGER);
     #JAN072021-IBM IIB REST_GET_SSL_DB you can send data via the header and use the GET operation
```

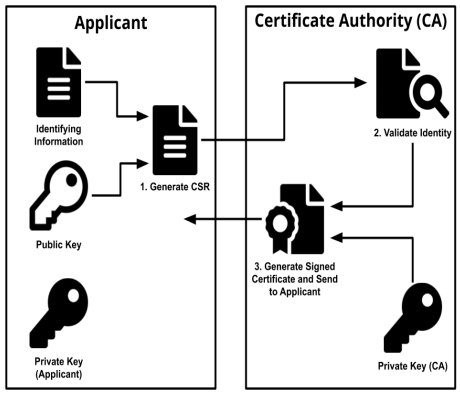
You can add a Request schema via the model definition in the REST API editor then you can add one, two and three elements as string. You can also use Parse Query String property in the HTTP Input node. If an HTTP request to http://localhost:7800/MessageFlow1/a%20space/b/c, the request is routed to the message flow that is deployed with the HTTPInput node, and the path segments are be placed in the local environment tree. JAN072021-IBM IIB REST_GET_SSL_DB 11:46

But the problem with the specification is that the keys and values, even the number of parameters in your JSON request are not validated even though your model specifically said the parameters have a certain case, are required and are type string. Only thing that is validated is that JSON is proper JSON format

HTTPS webservice

JAN072021-IBM IIB REST_GET_SSL_DB 20:50. IIB you can either consume HTTPS webservice or expose it. HTTPS is a transport layer security. When you are exposing the service, you are the provider. You will have a keystore. When you are consuming you are the client, and you will have a truststore.
In ikeyman create a keystore of type JKS called tstore.jks.

(JKS, Java Key Store. You can find this file at sun.security.provider.JavaKeyStore. This keystore is Java specific, it usually has an extension of jks. This type of keystore can contain private keys and certificates, but it cannot be used to store secret keys. Since it's a Java specific keystore, so it cannot be used in other programming languages. The private keys stored in JKS cannot be extracted in Java.)
The tstore.jks is protected by a password. If we are planning to use tstore.jks as a keystore (since we are the provider) then we choose personal certificate from the ikeyman dropdown. If we are using the file as a truststore (since we are consuming a service) we choose Signer certificates.



Java and Java Compute Node

JAN082021-IIB ExtJavaCall_DbNode_MQReplyPubSub.

```
--Creating Procedure For Calling External_Java_File(REGSTR.java), method RECORDS CREATE PROCEDURE EXJAVACALL(IN PRO1C INTEGER,IN PRO2C
   INTEGER,IN VNAME CHARACTER) RETURNS CHARACTER LANGUAGE JAVA EXTERNAL NAME "com.mss.REGSTR.RECORDS";
```

IIB 10 uses IBM java which includes oracle java's 1.7 features. IBM Integration Bus v10 ships with Java 8 in IIB 10.0.0.11 and later fix packs on all supported distributed platforms except Solaris and HP. However, the IIB Toolkit is unable to compile Java source code containing Java 8 language constructs such as Functional Interfaces, default and static methods in Interfaces and Lambda Expressions. If you try to deploy a bar file which contains or references Java 8 compiled class files to an IIB node running 10.0.0.10 or earlier then you may see one of the following errors: JavaCompute node with an onInitialize method BIP4157E: The user-defined node 'Java Compute' could not be deployed. Details: java.lang.UnsupportedClassVersionError: JVMCFRE003 bad major version class=Java18Test, offset=6.
MbJavaComputeNode is the superclass of all the classes used in JCN.

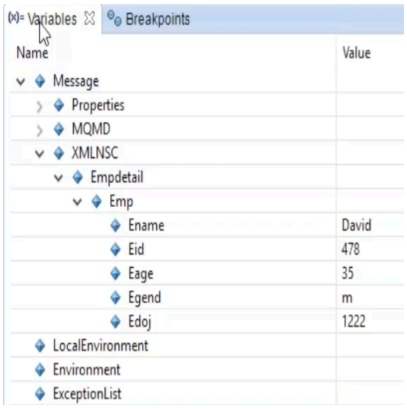
JAN182021-IIB JCN_MessSet The Filtering message class in JCN wizard is not aimed for transformation. It can't construct the new message. It is used to check the data without changing it like a filter node, then apply filtering rules.
JCN template for Java Architecture for XML Binding (JAXB) class relates to marshalling and unmarshalling. Java classes are generated from an XSD. You can create new XML data using the constructor’s classes and methods. You can also read newly created XML data.

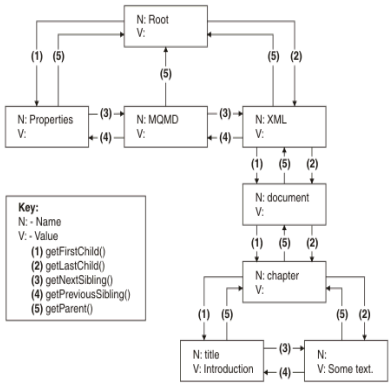
When you have finished the template wizard you will see extends MbJavaComputeNode. This superclass is an abstract class. It has abstract methods and concrete methods. Abstract methods that are already implemented can be called in our JCN. With concrete methods are not implemented so we need to implement the declaration. The evaluate method is a concrete method that we need to implement. The evaluate method is akin to the Main method in ESQL

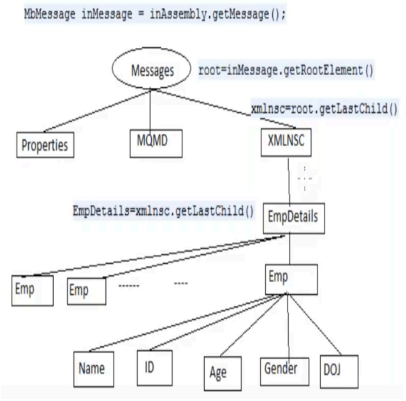
```
public void evaluate(MbMessageAssembly inAssembly) throws MbException {
   CREATE FUNCTION Main() RETURNS BOOLEAN
```

```
//input tree accessing //MbMessage root1 = inAssembly.getMessage(); MbElement root = inAssembly.getMessage().getRootElement(); //MbElement
   Body = inAssembly.getMessage().getRootElement().getLastChild().getLastChild(); //String jlKt=""; //Outputtree pointer MbElement xmlnsc
   = outAssembly.getMessage().getRootElement().getLastChild(); //INPUT MbElement
   empDetails=root.getLastChild().getLastChild().getFirstChild(); //MbElement empDetails=root.getLastChild() String
   SName=empDetails.getFirstChild().getValueAsString(); String SmpID=empDetails.getFirstChild().getNextSibling().getValueAsString();
   String SmpAge=empDetails.getFirstChild().getNextSibling().getNextSibling().getValueAsString();
```

The MbMessageAssembly is capable of holding the entire logical tree. The logical tree consists of message tree (properties, MQMD, XMLNSC), LocalEnvironment, Environment and ExceptionList







Connect DB JAVA type 4 driver

JAN072021-IBM IIB REST_GET_SSL_DB 32 minutes. Before we connected using the ODBC driver which is an operating system dependent driver. This is a pure java driver that connects using the Java API calls. The DB vendors like IBM and Oracle will provide the JAR files to connect to their databases. In IIB you need to enable some settings in the Configurable Services JAN192021-IIB JCN_ForLoop&JDBC&ModellingPt1 8 minutes. You can make changes that will be registered to the broker (right click broker in the toolkit, then select 'Start Web User interface'). Use the "mqsireportproperties brokerName -c AllTypes -c AllReportableEntityNames -r". From the output you can see the different connectionUrlFormat for DB2, Oracle, MySQL etc.
connectionUrlFormat='jdbc:db2://[serverName]:[portNumber]/[databaseName]:user=[user];password=[password];' connectionUrlFormat='jdbc:oracle:thin:[user]/[password]@[serverName]:[portNumber]:[connectionUrlFormatAttr1]'
JDBCProviders is the Configurable service that we need to Configure. We also need type4DriverClassName='com.ibm.db2.jcc.DB2Driver' which we will reference in the code. This class is available in the DB2 JAR file.

You should also tick the exception option in the properties. For example if you are using HTTP then the response will contain the error.

BIP2322E: Database error: SQL State ''IM002''; Native Error Code '0'; Error Text ''[unixODBC][Driver Manager]Data source name not found and no default driver specified

Design, requirements gathering

What questions will you ask? How will you implement these? For example, if you want to expose a web service what kind of notes will you take?

MQRFH2 Tree

The MQRFH2 header is used to pass messages to and from an integration node that belongs to IBM® Integration Bus. In a message, the MQRFH2 header follows the WebSphere® MQ message descriptor (MQMD) and precedes the message body, if present.

What is the use of MQRFH2 header? You can pass application data to other flows if the protocol is MQ. In the example below the MQ Output puts a message to a queue called test2. That then triggers another message flow called SecondFlow.msgflow which has an MQ Input to receive the message. You cannot use Environment.Variables.orderPerson when traversing from FirstFlow.msgflow to SecondFlow.msgflow. So how can you pass the data in Environment.Variables.orderPerson?

In the flow below the environmental variable is not available for the “call Service” but is accessible from the compute nodes. The ‘call Service’ contains:

```
SET OutputRoot.XMLNSC.ServiceResponse = InputRootXMLNSC.shiporder.shipto;


In the first flow: SET Environment.Variables.orderPerson = InputRoot.XMLNSC.shiporder.orderperson;
```

This variable is then needed in the second flow:

```
SET OutputRoot.MQRFH2.usr.oPerson = Environment.Variables.orderPerson;
```

Since the Compute1 node has an Out1 → HTTP Reply and Out → MQ Output you need to construct a message for the tree that will go to HTTP Reply & a different message tree that goes to MQ Output.

```
CALL CopyMessageHeaders();
DELETE FIELD OutputRoot.HTPResponseHeader;
SET OutputRoot.XMLNSC.Acknowledgement = ‘Sent to Downstream’;
PROPAGATE TO TERMINAL ‘out1’; --this goes to the HTTP Reply which sends a response to the client that initiated the HTTP request
```

(FYI if you added DELETE NONE to the last ESQL it will not delete the Message Assembly/Logical Tree after propagating)

---------- Second message assembly being constructed from scratch again

```
SET OutputRoot.Properties = InputRoot.Properties;
SET OutputRoot.MQMD = InputRoot.MQMD;
DELETE FIELD OutputRoot.HTPResponseHeader;
SET OutputRoot.MQRFH2.usr.oPerson = Environment.Variables.orderPerson;
SET OutputRoot.MQRFH2.usr.shipto = InputRoot.XMLNSC.ServiceResponse; – this is the response from ‘call Service’. This Input Assembly was object was constructed in the ‘call Service’.
RETURN TRUE; – by default means it will propagate to the OUT terminal so no need for an explicit propagate.
```

Once the 2nd message assembly reaches the 2nd flow via MQ protocol a new Output message assembly is created using the MQRFH2 header values.

Publication Node

This node handles publishing to the Topic within MQ. MQ can contain subscriptions that will listen to what is published if the topic string matches.

Shared vs Static Library.

30DEC2020-IIB RoutingTryCatchTrace 17 minutes. What is the difference between a Static and Shared library in IBM Integration Bus? Libraries can't include message flows only subflows. The subflows could take care of say error handling. This is common to many applications. If you deploy an application that references a shared library you see this error BIP1301E: the application cannot be deployed since it references a shared library that has not been deployed to the Integration Server (drag and drop to the Integration Server). But a static library will not complain since it is included in the deployed application.

- A. Shared libraries may include additional jar files while Static libraries cannot.
- B. Shared libraries can be deployed to an Integration Server while Static libraries cannot.
- C. Shared libraries encapsulate common code that may be used by multiple applications while Static libraries do not.
- D. Multiple applications can reference the same Shared library without having to include the library as part of the build as in the case of Static library.
- E. Static library appears under Included Libraries folder. Shared libraries appear under Referenced folder.

Compile BAR inline

15 minutes 04JAN2021-IIBFileNodeBARMQReplyTryCatchFlowOrder
To include message flows as compiled message flow (.cmf) files, and to include ESQL code directly in the .cmf file of each message flow that references an ESQL file, select Compile and in-line resources. By default, when you add a message flow to a BAR file, it is added as a .msgflow file. By default, each ESQL file that is referenced by one or more of your message flows is deployed as an individual resource, and can be accessed by multiple .msgflow files. If any of the flows that you add to your BAR file contain a subflow that is defined in a .msgflow file, you must select Compile and in-line resources.

BAR override

Notes from Udemy Section 12.
Under Independent resources -> GeneratedBarFiles you can copy the bar file to a folder or run mqsireadbar (04JAN2021-IIBFileNodeBARMQReplyTryCatchFlowOrder 20 minutes).

After extracting the file you can copy lines from the META-INF/broker.xml and create different property files for different environments.

The line from the broker.xml below:

<ConfigurableProperty override=”HELLO” uri=”ExternalVariable#EXTVAR”/>

Can be copied to a new file called DEV.properties

ExternalVariable#EXTVAR = diffvalue

Then you can run mqsiapplybaroverride -b <path to desktop>/MAP_PRO.generated.bar -p DEV.properties -k MAP_PRO

Then go to <path to desktop>/MAP_PRO.generated.bar, copy it, go to the Toolkit, right click on the MAP_PRO application then, paste it. This results in a new BAR folder under the application that contains the new generated BAR file. Click on the new BAR and you will see the new properties in the RHS.

You can then run this command to deploy the BAR with the new properties.

$> mqsideploy BROKERNAME -e EG_NAME -a <path to desktop>/MAP_PRO.generated.bar

Global Cache

One of the most important interview questions. Udemy 17

Shared variables used to store a temporary variable in Cache available to all message flows in the same broker schema.
Supposing you want variables to be accessible across schemas. Clearly Shared variables will not help you.
Global Cache allows you to share variables across schemas AND execution groups or Integration node (broker).

mqsireportproperties IIBGURU -b cachemanager -o CacheManager -r

```
CacheManager
  uuid='CacheManager'
  policy='disabled'
  portRange='2840-2859'
  listenerHost=''
  shutdownMode='fast'
  objectGridCustomFile=''
  deploymentPolicyCustomFile=''

BIP8071I: Successful command completion.
```

mqsichangeproperties IIBGURU -b cachemanager -o CacheManager -n policy,portRange,listenerHost -v default,generate,localhost

This command will only allow variables to be shared within the Integration node (broker) only.

Two concepts you need to be aware of. Catalog server and container server.

Suppose I deploy an application in the default Integration server (EG) that will LOAD a variable in Global Cache.
Note that default is a primary server that contains all the Global Cache variables AKA a Catalog server.
Imagine that on EG2 I deploy an application that will RETRIEVE a variable in Global Cache.
EG2 is a container server meaning it will load a local copy of the Global Cache held on the primary EG namely default. If you have 2 Integration Servers acting as catalog servers then you have some form of high availability

If I run ‘mqsireload default’ then the values present in my Global Cache will be gone because the Global Cache will be reloaded.
This will also happen if I restart the broker.

Enabling Global Cache for multiple Integration Nodes (brokers)

You need a policy.xml file that will contain the names of the nodes. There are some samples under ~/iib-10.0.0.22/server/sample/globalcache.

You have to run the command for all brokers.

mqsichangebroker BROKER1 -b <PATH>/policy_two_brokers.xml
mqsichangebroker BROKER2 -b <PATH>/policy_two_brokers.xml

mqsichangebroker BROKER2 -b disabled //This will switch it off

---
A static method means that it can be accessed without creating an object of the class, unlike public
We might have different EG accessing an XML in Global Cache. We will need an ESQL code to invoke some JAVA code.
mqbrkruser@SATELLITE-L50D-B:~$ cat IBM/IIBT10/workspace/GlobalCacheApp/LoadGlobalCache/SubFlow/LoadGlobalCache_LoadCache.esql

```
BROKER SCHEMA LoadGlobalCache.SubFlow
CREATE COMPUTE MODULE LoadGlobalCache_LoadCache
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
DECLARE InBlob BLOB;
SET InBlob = ASBITSTREAM(InputRoot.XMLNSC, InputRoot.Properties.Encoding, InputRoot.Properties.CodedCharSetId); -- Converts XML to BLOB so
   that it can be added to the Global Cache
CALL LoadXMLToCache('GBKey', InBlob);
SET OutputRoot.XMLNSC.Message.Value='Successfully Loaded';
RETURN TRUE;
END;
CREATE PROCEDURE LoadXMLToCache (IN key CHARACTER, IN Value BLOB )
LANGUAGE JAVA
EXTERNAL NAME "GlobalCacheClass.LoadXMLToCache";
END MODULE;
```

mqbrkruser@SATELLITE-L50D-B:~$ cat IBM/IIBT10/workspace/GlobalCache_PRJ/src/GlobalCacheClass.java

```
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbGlobalMap;
public class GlobalCacheClass {
public static String MapName = "GBMAP";
public static void LoadXMLToCache(String key, byte[] xml) {
try {
// getGlobalMap is a static method. Static methods can be called without creating objects MbGlobalMap map =
   MbGlobalMap.getGlobalMap(MapName);
if(map.containsKey(key)){
map.update(key, xml);
}
else {
map.put(key, xml);
}
} catch (MbException e) {
e.printStackTrace();
}
}
public static byte[] GetCacheValue(String key) {
byte[] xml = null;
try {
MbGlobalMap map = MbGlobalMap.getGlobalMap(MapName);
xml = (byte[])map.get(key);
} catch (MbException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
return xml;
}
}
```

You can see the Global Cache in the Activity log, within the Web interface. Udemy 19

Opaque Parsing

In your FileNodesExamples your File Input node has an Opaque elements search for //Opaque which means whenever the Opaque element exists (xpath notation). It does not parse the elements inside the Opaque parent. You will just see the child elements as a string value.

JCN code to access a message set

In google drive “[2021-01-27] IIBFixJCNMulti.mp4” you can access MSET using XPath.

```
MbElement root =
  inAssembly.getMessage().getRootElement().getLastChild().getLastChild();
MbElement emp1[]= root.getAllElementsByPath("*");

MbElement Root=outAssembly.getMessage().getRootElement().getFirstChild();
Root.getFirstElementByPath("./MessageSet").setValue("JCN_CSV_MessageSet");
Root.getFirstElementByPath("./MessageType").setValue("{}:SDET");
Root.getFirstElementByPath("./MessageFormat").setValue("Text_CSV");

```

Timeout Notification is controlled by Timeout Control

, so long as they both have the same unique identifier. You also need to set the operation mode of the Timeout Notification to Controlled, not Automatic. The Timeout Control node has a Request location property InputRoot.XMLNSC.EmpDetails.TimeoutRequest. Within the Timeout you will have an Action, Identifier, StartDate etc. 29DEC2020-IIB Timeout_LabelRouting_MultiQueues

Failure and Catch terminals

An Exception will roll back to first node (The exception will always be in the last child of the ExceptionList) then it will check if the catch exception is handled or not 21DEC2020-IIB FixHTTPProdCon_ESQLThrow 36 minutes. If it is then the Message and ExceptionList trees are passed to the node at the end of the catch terminal. The ExceptionList will contain the exception that will give you a better idea of what & where in the flow the problem happened. If there is an exception raised while processing downflow of the Catch then the exception will feedback then traverse the Failure node path only if Transaction Mode is Yes.
If catch is not connected it will check if Failure is connected. Then the default exception and NOT the exception we threw go to the node at the end of the Failure terminal only if Transaction Mode is Yes. The default exception will say you did not handle the Catch. The advantage of the Catch terminal is that you will get an exception which will give you a better idea of what & where in the flow the problem happened.

If the format of the input message is not correct, then the flow will traverse the Failure terminal only.

MQ Failure and Catch

When the message that is not handled goes back to the IN queue the message is considered a poison message as it will not allow new messages to be 'MQGET' by the flow until the poison message is removed. The poison message can be handled by designating another queue as the backout queue to the IN via MQ Explorer or giving the backout queue name in the flow or setting a DLQ.21DEC2020-IIB FixHTTPProdCon_ESQLThrow 36 minutes

Try Catch node

30DEC2020-IIB RoutingTryCatchTrace. This allows us to handle the exception further up the flow. The catch terminal will send the message and the ExceptionList. Your terminal goes to a Compute node to handle the exception.

Propagation

```
RETURN TRUE; -- If we forget this statement, the message will still propagate as if it was there to the OUT terminal


PROPAGATE;
RETURN FALSE; --will do the same.


PROPAGATE TO TERMINAL 'out' OR 0;
RETURN FALSE; --will do the same.
```

Event Monitoring

https://youtu.be/Be92ez0wtSA?si=zB1o5WWxInExQNJB Feature in IIB where we can log some events. These can be categorized as Transactional or Terminal (node terminal). For example, the HTTP Input node has a Monitoring tab under properties. This will allow the node to produce events as messages are processed. Events are published as XML messages that can be subscribed to by other IIB applications. The messages can capture message performance and stats.

You can map some key fields from the XML data to a custom app. This data is typically inserted into a database in PROD.

You first have to create an MQ queue called MONITOR.EVENTS.DEFAULT. Then create a topic called BROKER_EVENTS_DEFAULT with a Topic String $SYS/Broker/IIBGURU/Monitoring/IIBGURU_EX/<flowname>.

For an MQ pub/sub broker, the topic root is $SYS/Broker. For example,

$SYS/Broker/integrationNodeName/Status/ExecutionGroup/integrationServerName

The Topic String is where the event message is going to be put to. Instead of <flowname> you can enter the # symbol for all flows. Create an MQ subscription called MONITOR.EVENTS.DEFAULT and select the Topic you just made as well as the queue manager and the queue you just made since you want to receive these publications on an MQ queue.

Then configure and enable event monitoring on a message flow. Your event source could be a Transaction start on an HTTP Input node or the end on an HTTP Output node. This information allows us to capture and report message flow performance.

Activate broker monitoring events using:

```
$ mqsichangeflowmonitoring IIBGURU -e IIBGURU_EX -k EvalEmp -f EVAL.PROC.EvalProc -c active
BIP8071I: Successful command completion.
$ mqsireportflowmonitoring IIBGURU -e IIBGURU_EX -k EvalEmp -f EVAL.PROC.EvalProc
```

Since the flow is running on a broker that is linked to the queue manager the monitoring is set up. Now when you send a message through your EVAL.PROC.EvalProc flow the monitoring will be triggered to create an event message in XML format on the queue MONITOR.EVENTS.DEFAULT.

But before that you have to create a flow that will work with the event message. This flow has an MQInput node that has the queue name set to MONITOR.EVENTS.DEFAULT. Then the XML will be passed to the next node to insert into the database.

Set up HTTPS web service

You have to set up SSL in the JVM settings of the integration server via the ComIbmJVMManager object.

The ikeyman utility allows you to create a keystore database file of type JKS. Call it iibkeystore.jks and provide a password.
The keystore initially only contains a private key.

Click on new self-signed certificate with key label personalcert then select it then click on validate. The certificate is contained within the keystore. You then need to associate the keystore with the execution group.

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiWyMPF9qPvAhW1WxUIHSSSD_cQFjAAegQIBBAD&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2Fself-signed_certificate&usg=AOvVaw1ulmEXUANNETSV5MnkQghm

Commands to associate keystore with your execution group.

```
mqsichangeproperties IIBGURU -e IIBGURU_EX -o
    ComIbmCacheManager -n explicitlySetPort -v 7788
mqsichangeproperties IIBGURU -e IIBGURU_EX -o
    ComIbmJVMManager -n keystoreFile -v /home/mqbrkruser/iib-
    10.0.0.22/MiddlewareGuy/iibkeystore.jks
mqsichangeproperties IIBGURU -e IIBGURU_EX -o
    ComIbmJVMManager -n keystoreType -v JKS
mqsichangeproperties IIBGURU -e IIBGURU_EX -o
    ComIbmJVMManager -n keystorePass -v
    defaultKeystore::password

mqsisetdbparms IIBGURU -n defaultKeystore::password -u ignore
    -p G8n35hj1
mqsireload IIBGURU -e IIBGURU_EX
```

Validation settings for input type nodes

User Trace
(30DEC2020-IIB RoutingTryCatchTrace 28 minutes) Logs all validation failures to the user trace, even if you have not asked for user tracing of the message flow. Use this setting if you want processing of the message to continue regardless of validation failures.
mqsichangetrace IIBGURU -e IIBGURU_EX -u -l debugTree -c 999999
mqsireadlog IIBGURU -e IIBGURU_EX -u -o somename.xml
mqsichangetrace IIBGURU -e IIBGURU_EX -u -l none

Local Error Log
Logs all validation failures to the error log (for example, the Event Log on Windows). Use this setting if you want processing of the message to continue regardless of validation failures.

Exception
The default value. An exception is thrown on the first validation failure encountered. The resulting exception list is shown below. The failure is also logged in the user trace if you have asked for user tracing of the message flow, and validation stops. Use this setting if you want processing of the message to halt as soon as a failure is encountered.

Exception List
Throws an exception if validation failures are encountered, but only when the current parsing or writing operation has completed. The resulting exception list is shown below. Each failure is also logged in the user trace if you have asked for user tracing of the message flow, and validation stops. Use this setting if you want processing of the message to halt if a validation failure occurs, but you want to see the full list of failures encountered. This property is affected by the Parse Timing property; when partial parsing is selected the current parsing operation parses only a portion of an input message, so only the validation failures in that portion of the message are reported.

ASBITSTREAM in ESQL

Converts payload (e.g., XML) into bitstream. Bitstream means a BLOB. A BLOB is a string of hexadecimal characters. The benefit is that you can insert the payload into a database. Another reason is that you may want to cast an XML or JSON into a string.

```
SET InBlob = ASBITSTREAM(InputRoot.XMLNSC, InputRoot.Properties.Encoding, InputRoot.Properties.CodedCharSetId); -- Converts XML to BLOB so
   that it can be added to the Global Cache
```

```
--This statement will convert the XML or JSON tree to string.
SET XMLInput = COALESCE( CAST(ASBITSTREAM(InputRoot.XMLNSC CCSID 1208) as CHAR CCSID 1208), ''); SET JSONInput = COALESCE(
   CAST(ASBITSTREAM(InputRoot.JSON.Data CCSID 1208) as CHAR CCSID 1208), '');
```

Every language has a particular encoding that the operating system can understand. CCSID defines the characters that are allowed (1208 = UTF-8). If you omit the CodedCharSetId then it will still convert fine to a BLOB if your message is using English characters. However if the message contains something unusual like some Cyrillic characters then without CCSID it won't know how to handle the characters.
https://youtu.be/bKVmhaRqsCg?si=RB3ppXk_-MWnaoMy Encoding 273 corresponds to Unix (Non-Intel) operating systems, for example AIX or Solaris Spark, this is referred to as Big Endian. Encoding 546 corresponds to Linux and Windows operating systems on Intel, this is referred to as Little Endian.

Aggregation

https://iteritory.com/ibm-integration-bus-iib-aggregate-nodes-sample-with-http-web-services/
The following JSON message is sent to the REST API /aggregationcustomerapi/v1/customers/.

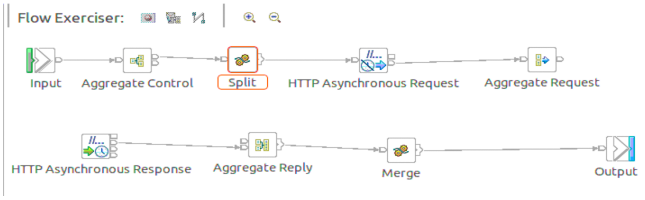
{

"CustomerInfoReq": [
{
"custId": "Amit"
},
{
"custId": "Shreya"
},
{
"custId": "Pete"
}
]
}

The request is sent to a sub flow.

```
SET OutputLocalEnvironment.Destination.HTTP.RequestURL = reqURL || myref.Item[I].custId;
                  PROPAGATE TO TERMINAL 'out' DELETE NONE;
                  SET I = I + 1;
          END WHILE;
          RETURN FALSE;
   END;
```

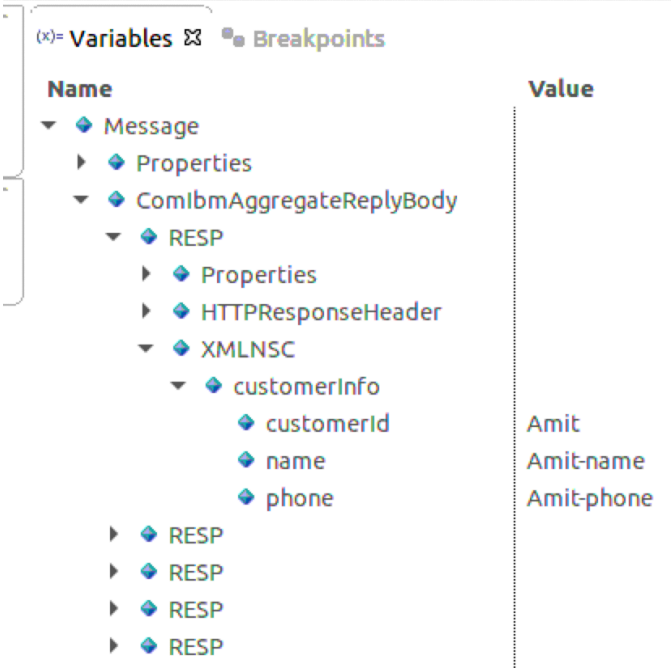
Within the Split compute node, the SET initializes the output buffer.



By default, the node clears the output message buffer and reclaims the memory when the PROPAGATE statement completes. The PROPAGATE … DELETE NONE will not delete the Message Assembly/Logical Tree after propagating so that the message is available for routing to the next destination. Use DELETE NONE if you want the downstream nodes to be able to see a single instance of output local environment message, and exception list trees.

Propagation is a synchronous process. That is, the next statement (incrementing the counter in our example) is not executed until all the processing of the message in downstream nodes has completed. So that means the counter 'I' will be incremented after the http://localhost:7080/legacybackendservice/v1/customer/Amit web service response is sent to the AggregateReply.
After the while loop completes the return false command is run. This exits the code immediately. Only now will the flow pass on to the AggregateReply node.

The AggregateReply node creates a folder in the combined message tree below Root, called ComIbmAggregateReplyBody. Below this folder, the node creates a number of subfolders using the names that you set in the AggregateRequest nodes. These subfolders are populated with all associated reply messages.



AggregateControl, AggregateRequest nodes (Used in Fan-Out to broadcast the request message to multiple destinations). The request node records the number of requests.

AggregateReply (Used in Fan-In to collect responses) – checks if all responses are collected.

'Aggregate Name' property of AggregateControl & AggregateReply nodes should be the same. Our Aggregate Control & AggregateReply nodes have an Aggregate name AGGR.
https://ibmintegrationbus.wordpress.com/2019/06/14/aggregation-nodes/

The 'Folder Name' property of the AggregateRequest node decide how the input will be structured in Fan-Out flow. We are using RESP. RESP will be an array for multiple responses.

It combines the generation and concurrent fan-out of a number of related requests with the fan-in of the corresponding replies, and compiles those replies into a single aggregated reply message. IIB as a product makes it easy to implement complex integration scenario with Aggregation support.

FanOut flows: AggregateRequest and AggregateControl (whenever we use Aggregation control, we must use Aggregation request)
FanIn flow: AggregateReply (it will club the incoming multiple responses) from AggregateControl node and AggregateRequest node.

Reference: https://community.ibm.com/community/user/integration/viewdocument/ibm-integration-bus-for-developers?CommunityKey=77544459-9fda-40da-ae0b-fc8c76f0ce18&tab=librarydocuments