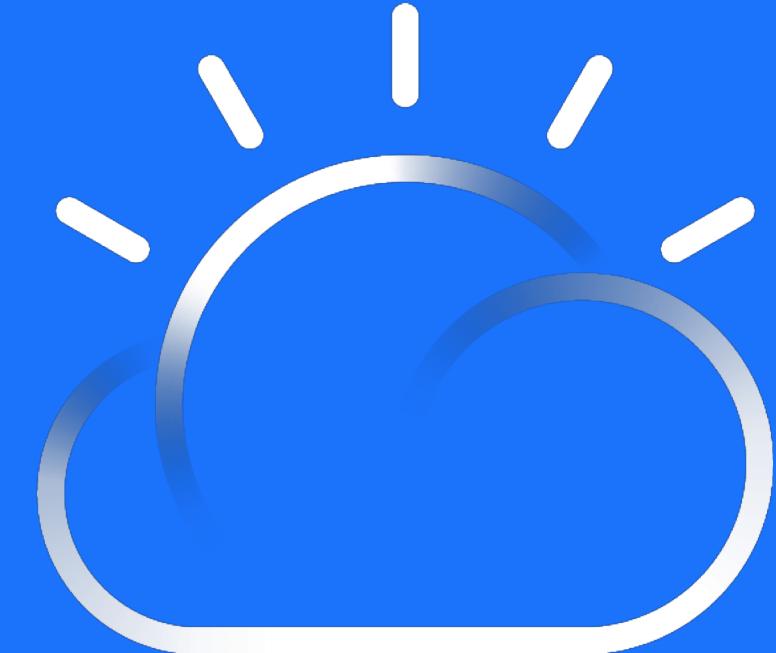


A07 Application Integration

A07 Becoming an
Integration ACE
with ACEv11



Sanjay Nagchowdhury
sanjay_nagchowdhury@uk.ibm.com

IBM Cloud

IBM

Agenda



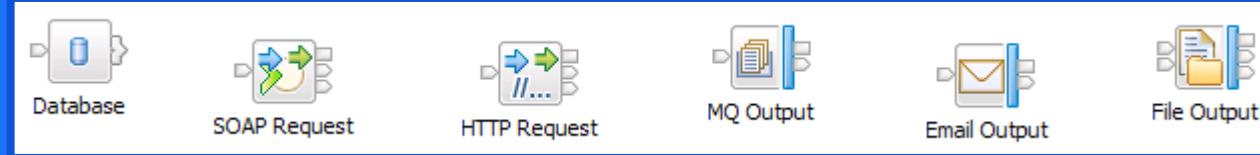
- Message Flow Design
- Test Considerations
- Demo Time!!
- Performance Considerations
- Transformation Options

Message Flow Design

Your connectivity solution

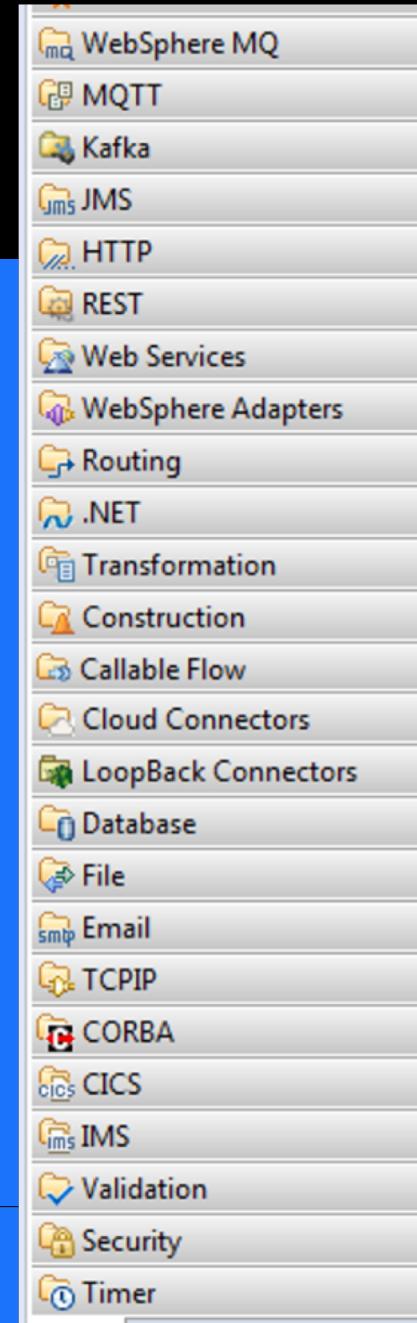
Built-in nodes encapsulate transports, technologies and applications

- Our intent is always to make the common tasks easy, and the rest possible!
- Use the built-in nodes to reduce the amount of custom code required
- This makes best use of the built-in facilities like activity log and resource statistics



We have built-in nodes to serve different purposes:

- Connecting with client applications
- Routing messages
- Transforming and enriching messages
- Processing events
- Handling errors in message flows



Developing an integration solution



IBM App Connect Enterprise provides different ways of creating your integration solutions.

- Using Applications and Libraries
- Using an Integration Service
- Using a REST API
- Using a Policy
- Using Patterns

Quick Starts

Start building your application with one of the following tasks.

[Start by exploring tutorials](#)

Use the **Tutorials** to learn more about the features in IBM App Connect Enterprise. [More...](#)

[Start by creating an application](#)

An **Application** is a container for all the resources that are required to create a solution. [More...](#)

[Start by creating an integration service](#)

An **Integration Service** is an application with a well-defined interface and structure. [More...](#)

[Start by creating a REST API](#)

A **REST API** is an application with a well-defined interface and structure.

[Start by creating a library](#)

A **Library** is a logical grouping of related code, data, or both. [More...](#)

[Start from WSDL and/or XSD files](#)

Use this task to create an **Integration Service**, **Application** or **Library** which includes your WSDL and/or XSD files.

[Start by discovering a service](#)

A **Service** allows you to invoke the remote system using discovered resources.

[Start by creating a Policy project](#)

A **Policy project** contains policies which control the behaviour of an **Integration Server**.

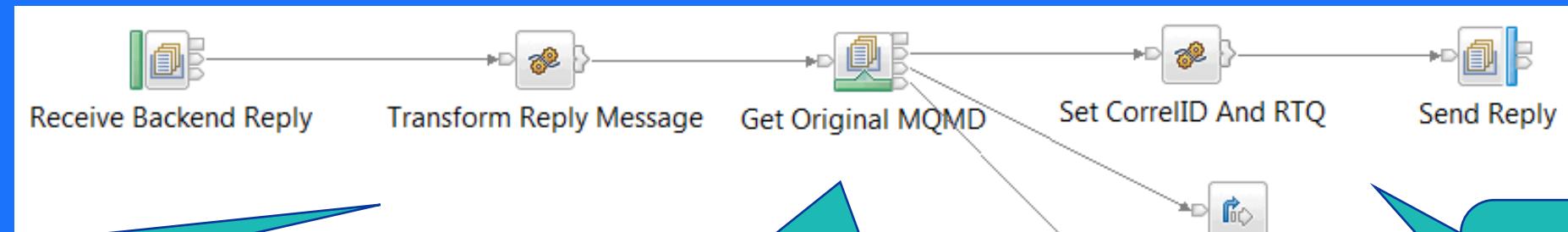
[Start from patterns](#)

A **Pattern** is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task. [More...](#)

Message Flow Structures



Asynchronous request-response message



Replies are correlated with the original requests.

Store the original request message id on a queue and then the reply message flow retrieves that message id information from the queue.

Handle MQGet Warning

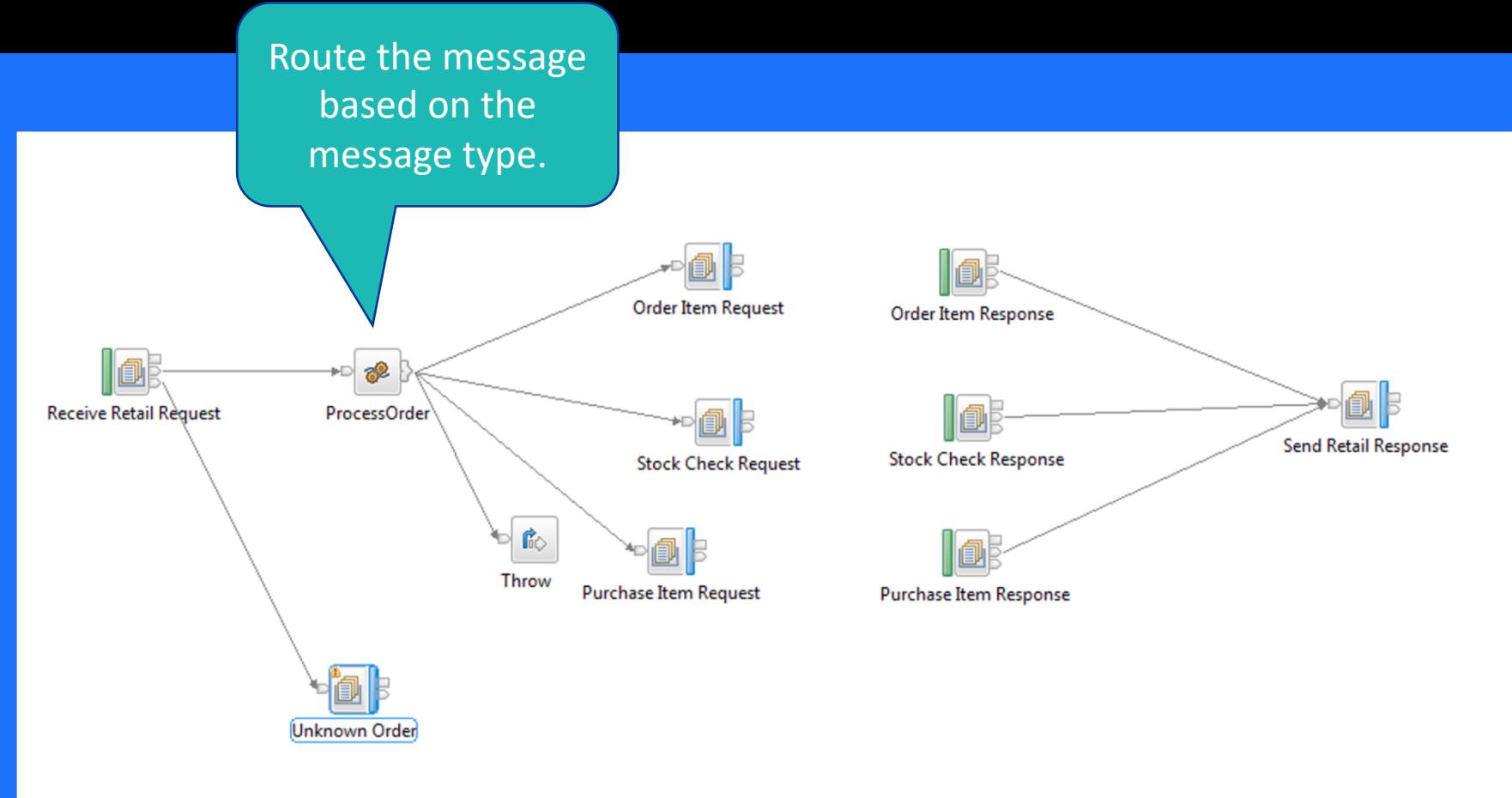
Handle Error No State Message

Use asynchronous flows to avoid bottlenecks.

Message Flow Structures

IBM.

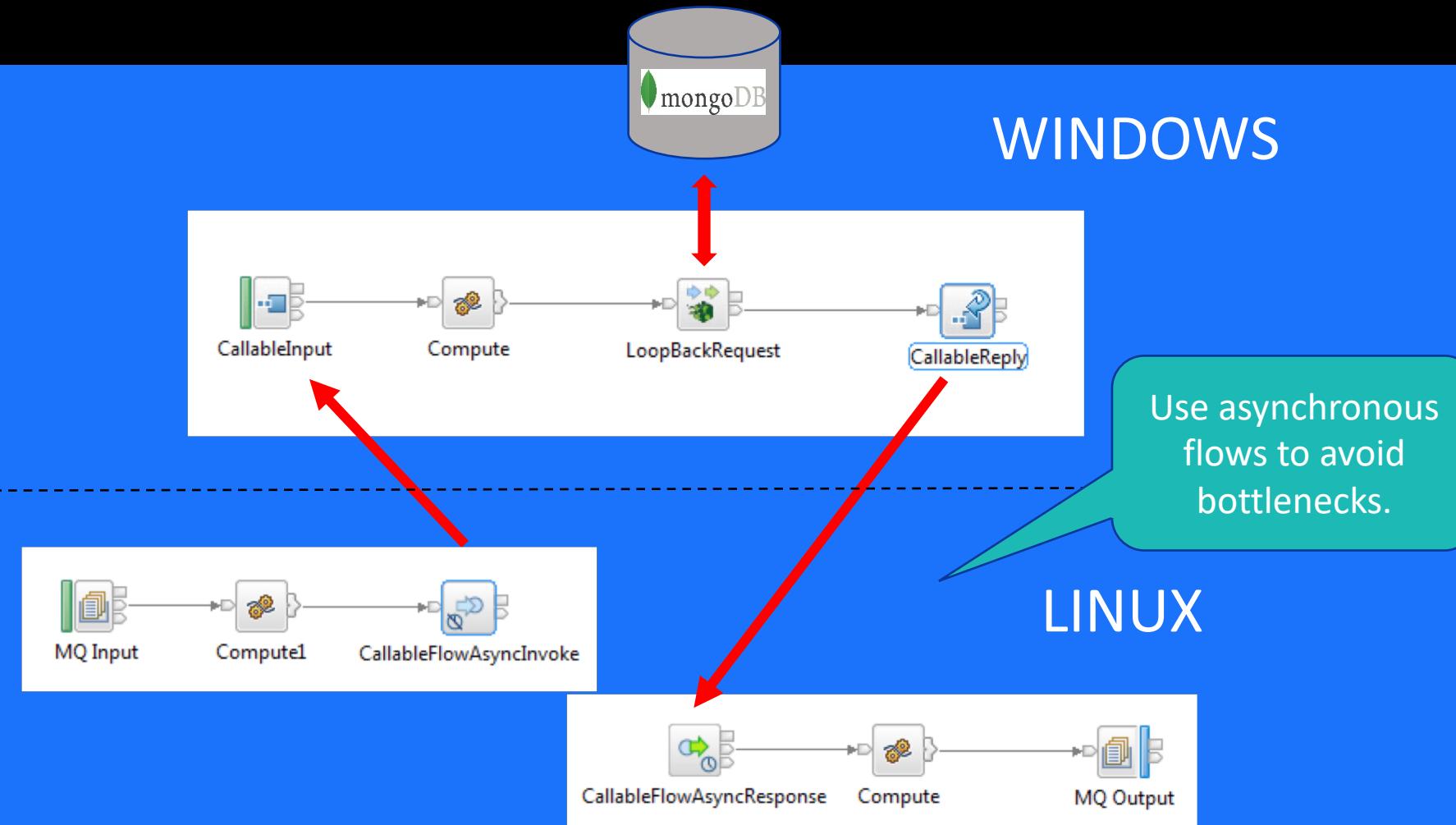
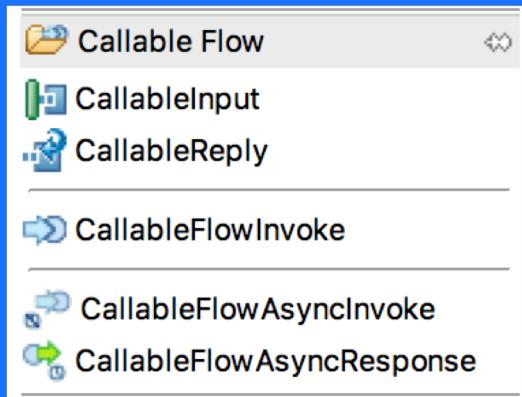
- Separation of processing
- Split out incoming different requests received at single entry point
- Find the message type upfront quickly and then route accordingly.



Message Flow Structures

IBM.

A message flow deployed to one integration server can directly invoke a message flow in a different integration server, passing the contents of the message body and the Environment folders.

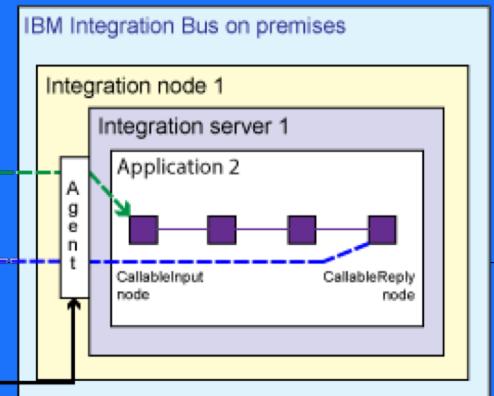
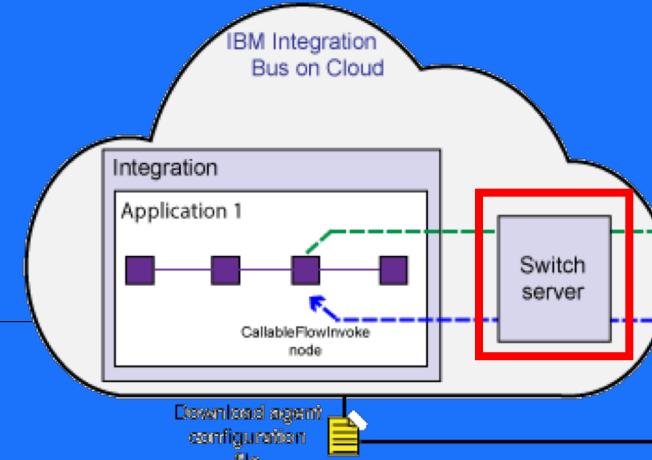
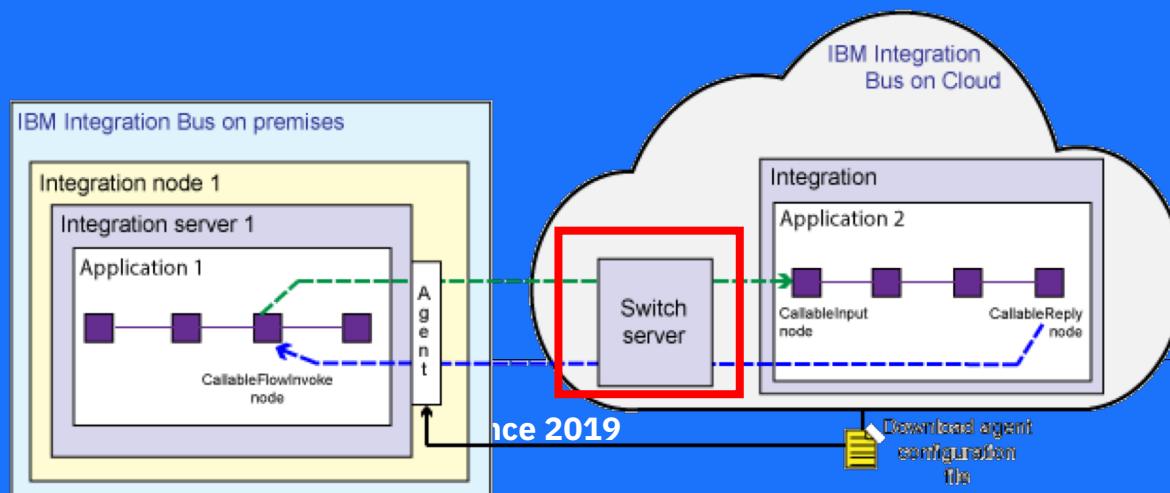
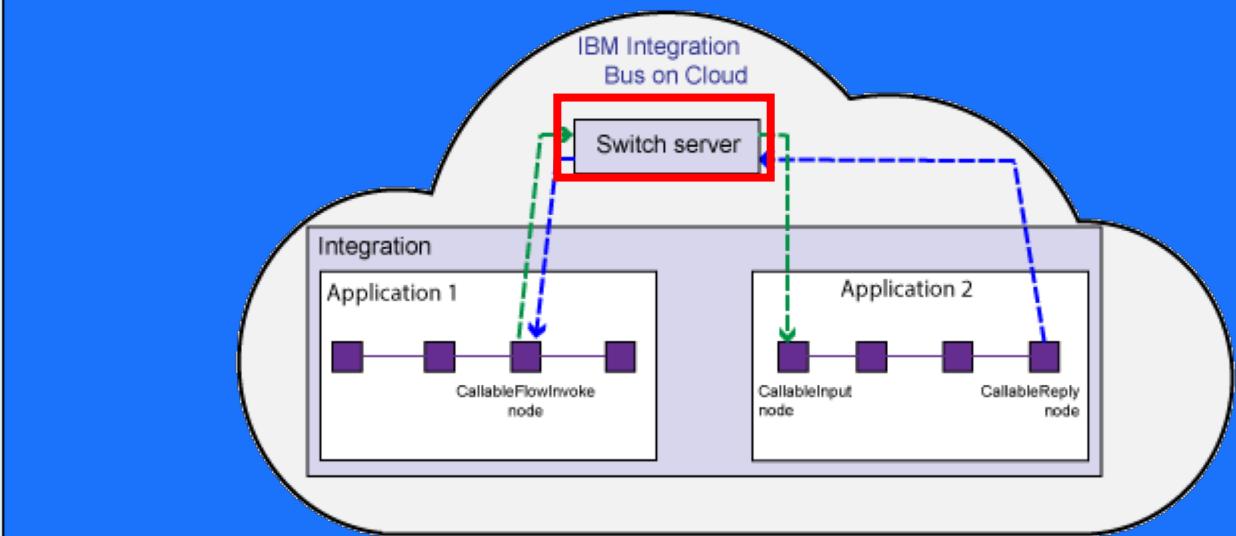
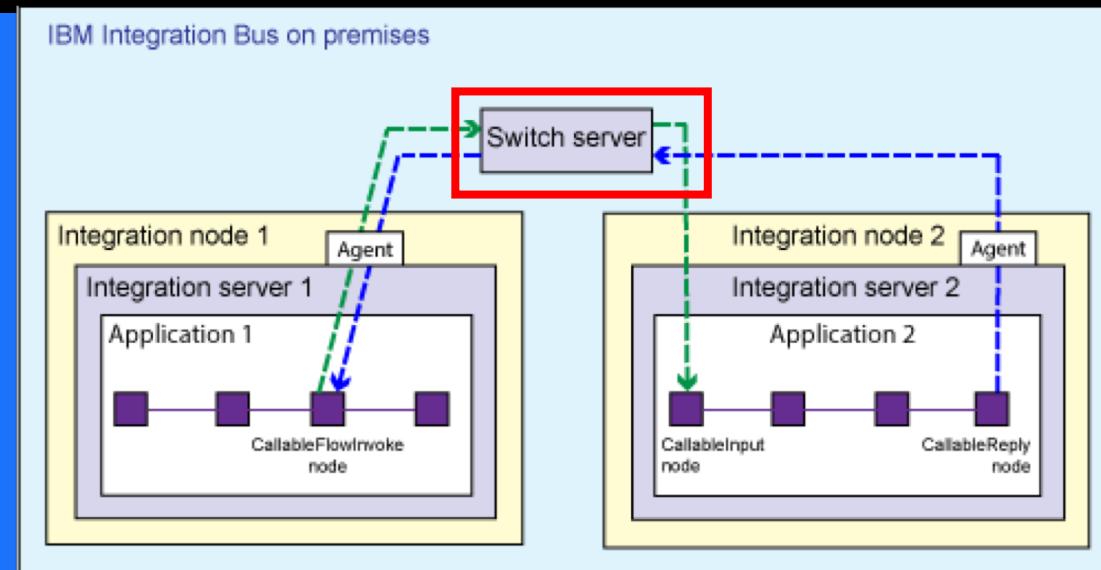


CallableInput Node Properties - CallableInput

Description	Endpoint Name*	UniqueEndpoint
Basic	Monitoring	

Message Flow Structures

IBM



Subflows

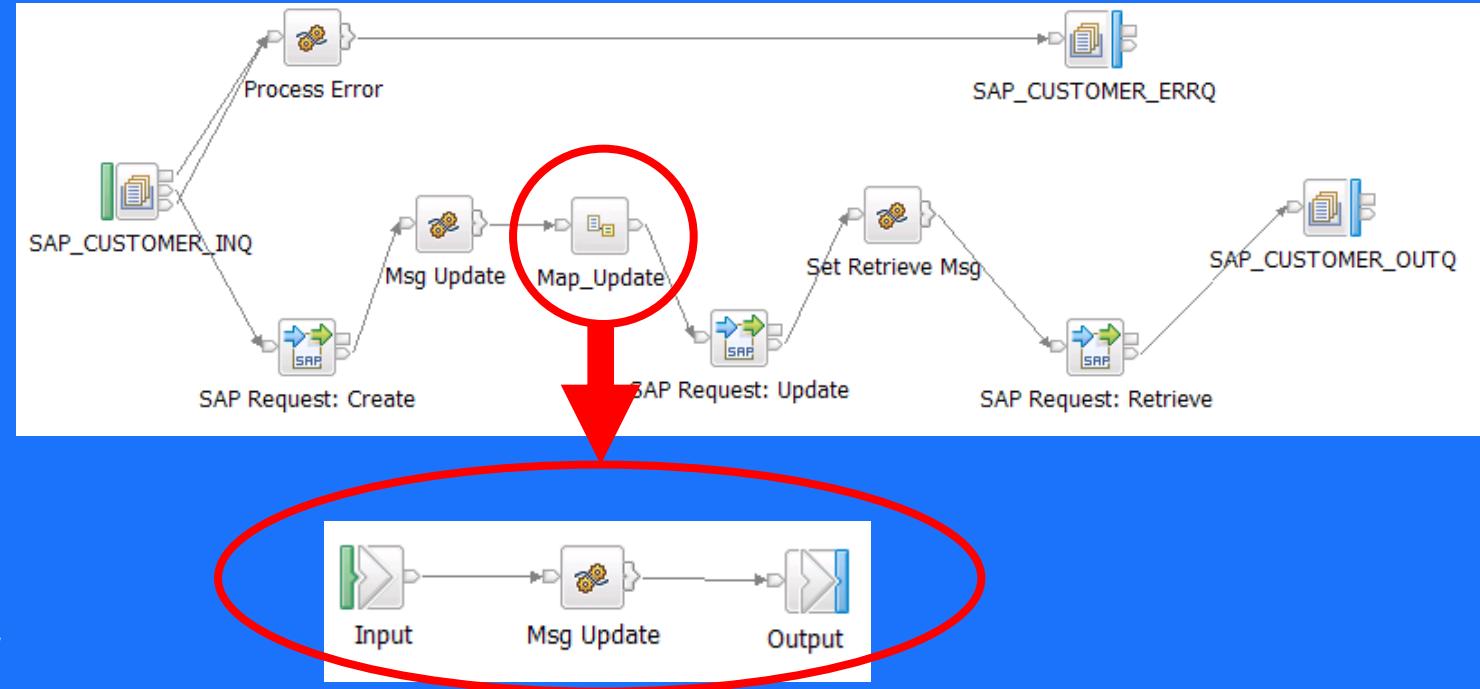
IBM

► Subflows are simply message flows that are invoked from another flow

- Input and output nodes in the subflow become terminals in the main flow
- Use subflows to break up large problems into smaller more manageable chunks

► Subflows are directly deployable to the runtime

- Shared subflows deployed just once per integration server
- Redeployment of a subflow is automatically picked up by any consumers



When it is Right Make a Pattern of It

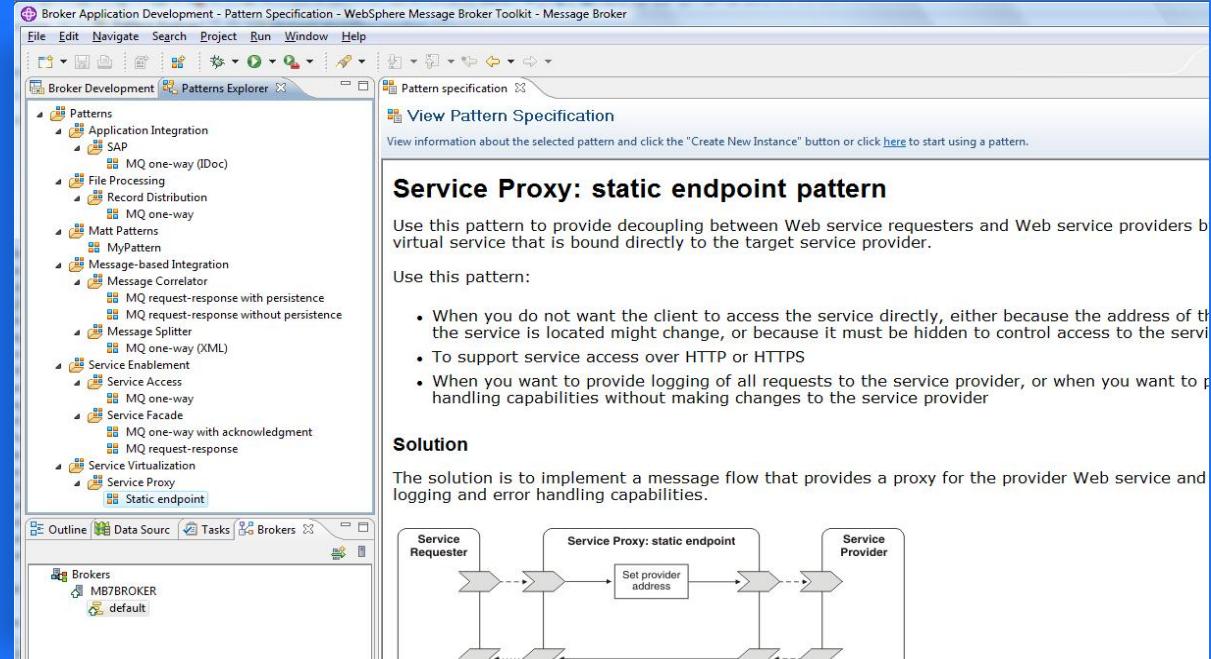


Patterns provide top-down, parameterized connectivity solution of common use cases

- e.g. Web Service façades, Message oriented processing, Queue to File...

They help describe best practice for efficient message flows

- Reduce common problems in flow development
- Establish best practice
- Reduces time to develop solutions
- Helps new message flow developers to come on board more quickly.

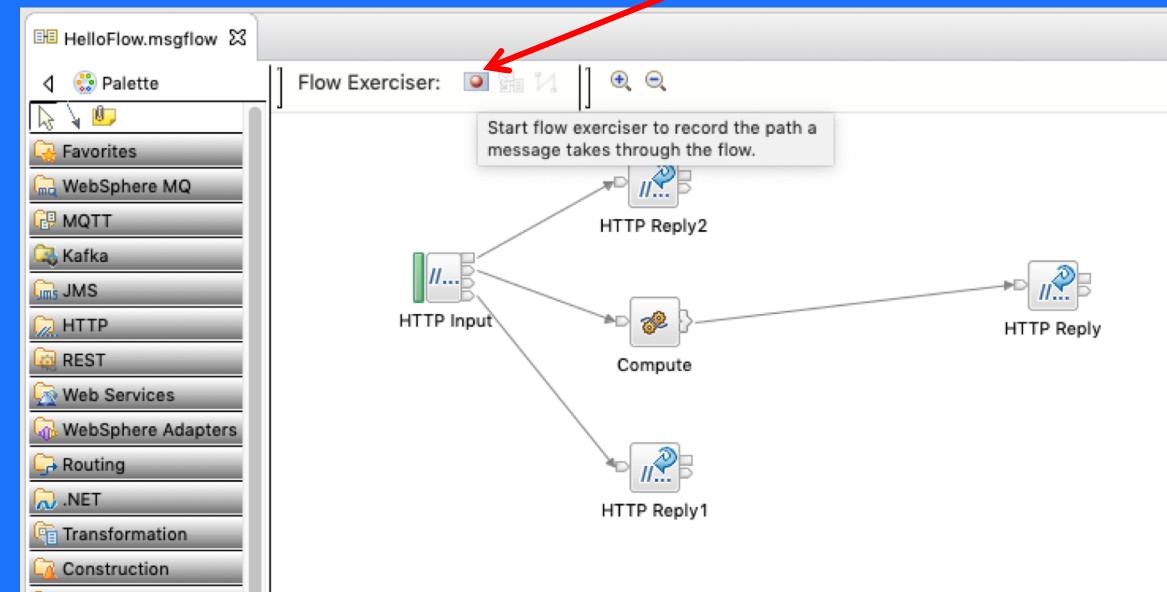


Test Considerations

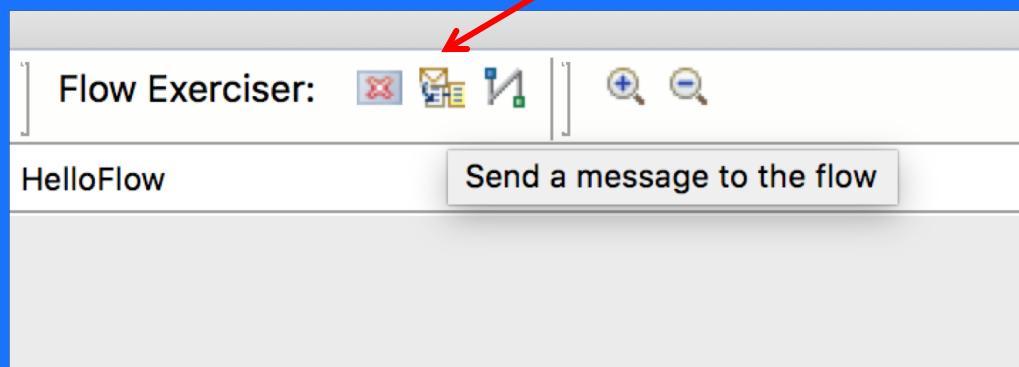
Flow Exerciser – use to test your flows

IBM

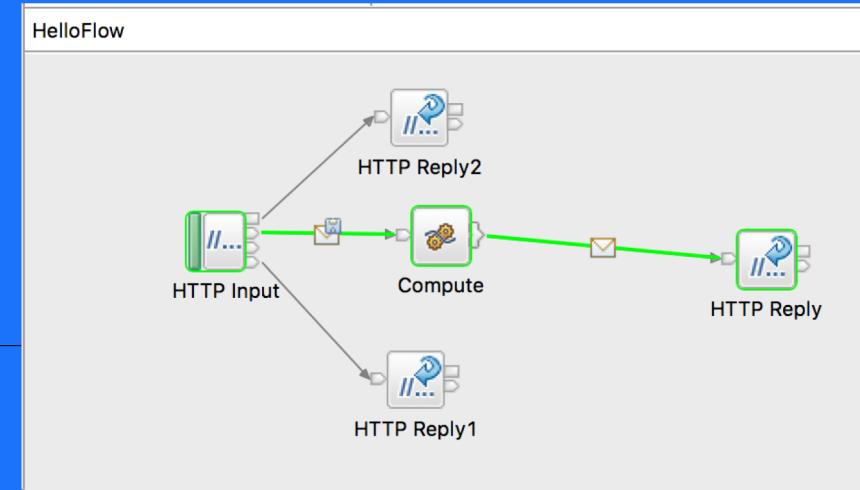
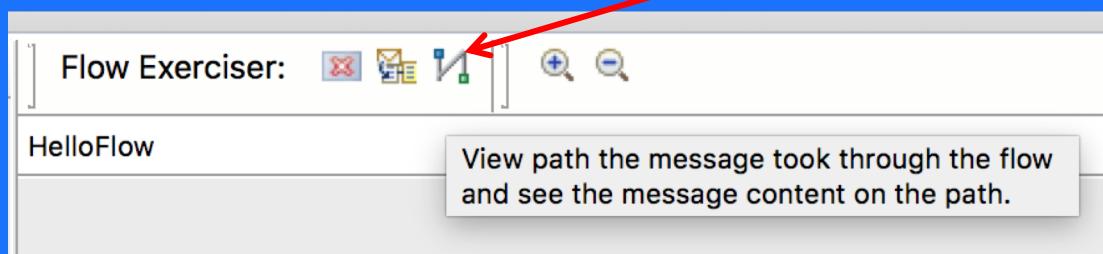
Recording: Start the flow exerciser



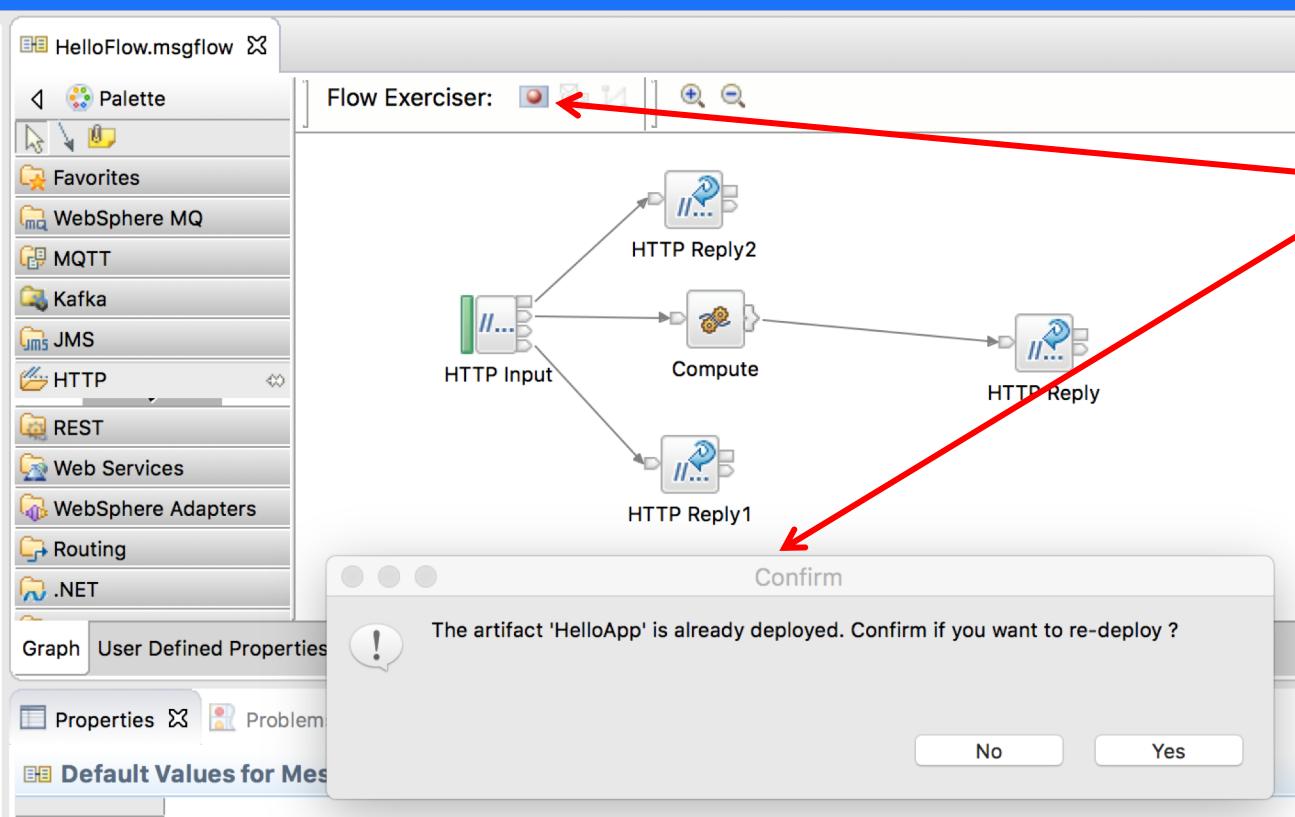
Injection: Send a message to the flow



Viewing: Viewing the message processing path and message content

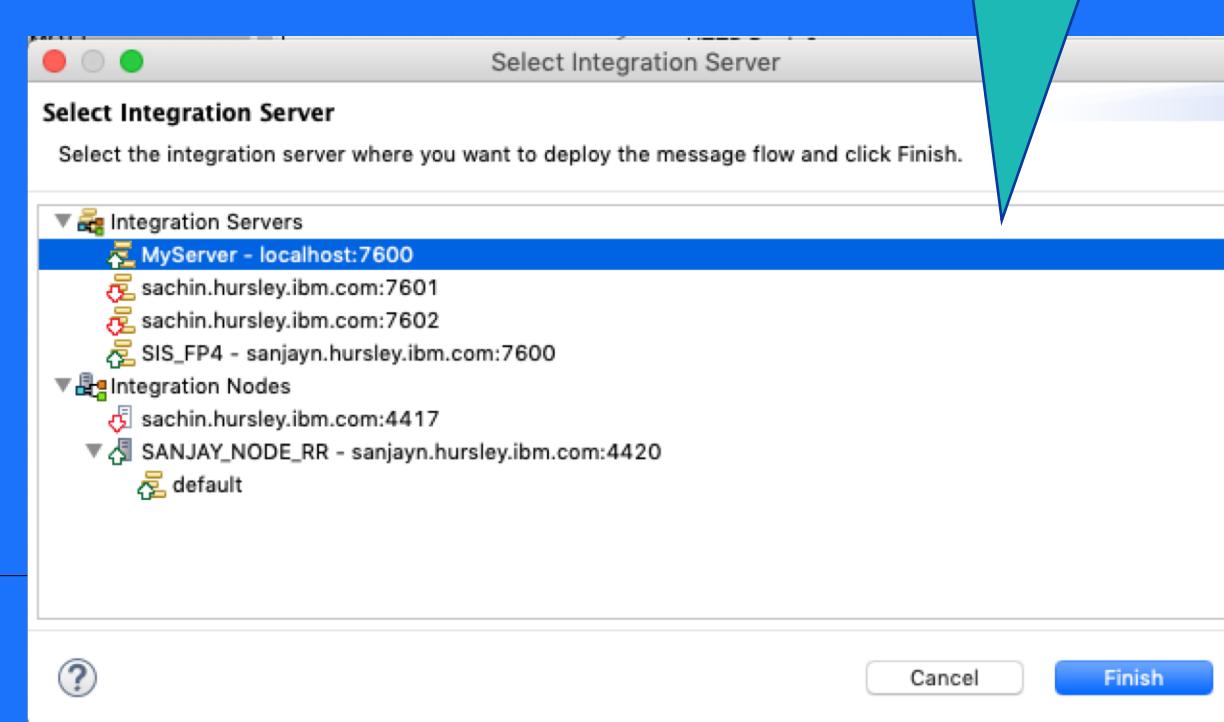


Flow Exerciser – recording a message

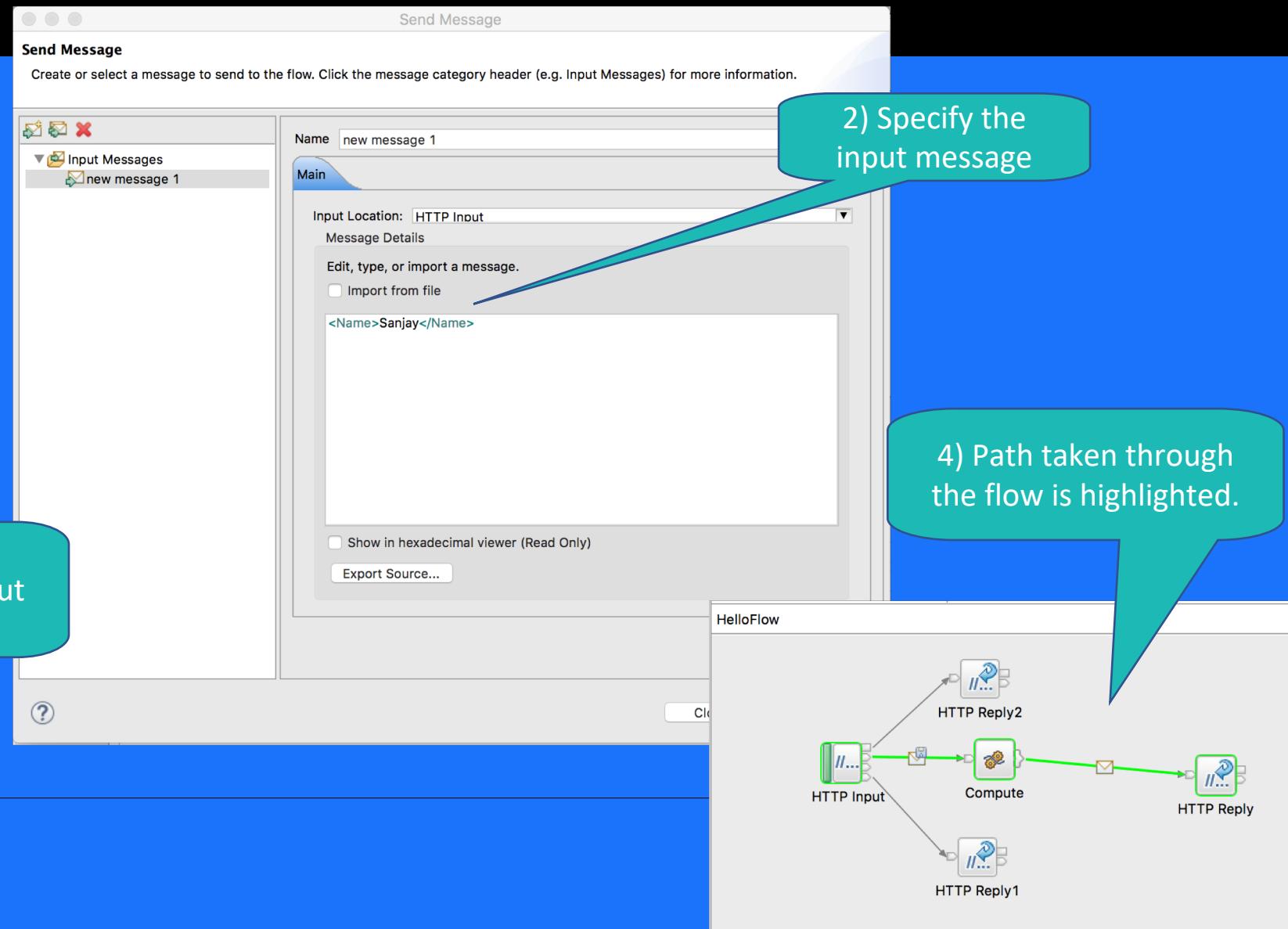
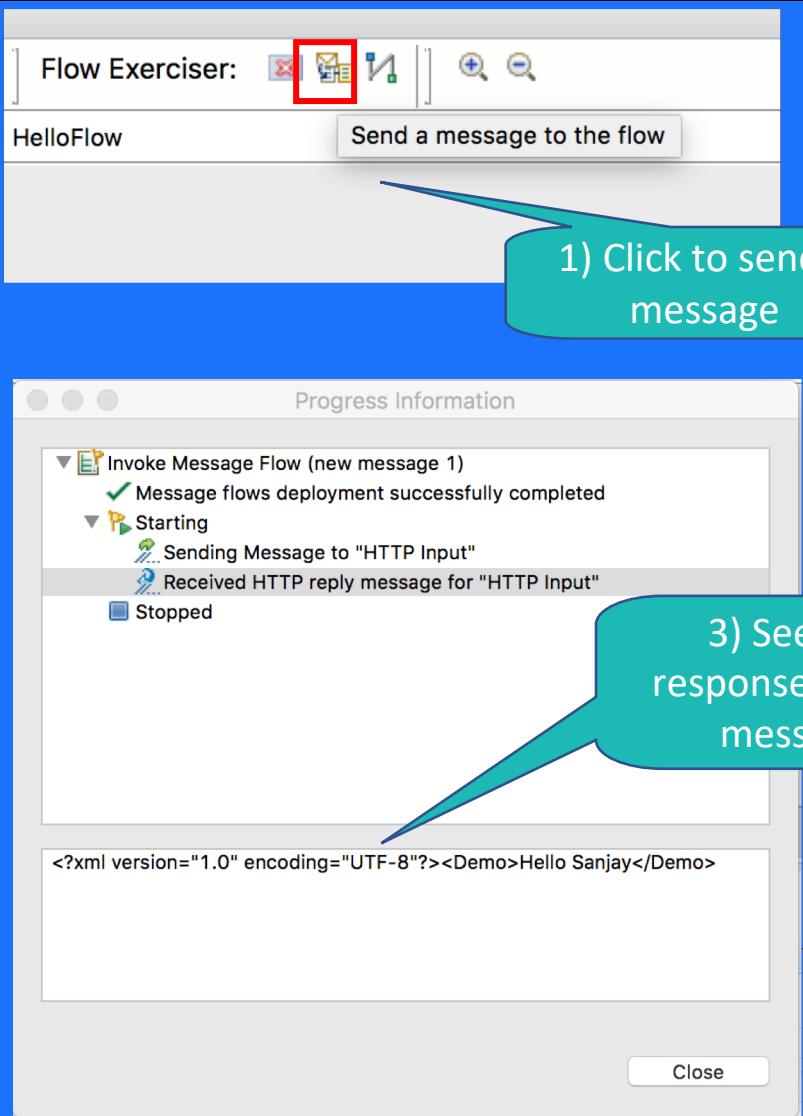


New in ACE V11, a confirmation dialog is shown to re-deploy if the Application is already deployed.

Choose a independent Integration Server or a node-owned Integration Server to deploy to.



Flow Exerciser – sending a message



Flow Exerciser – viewing a message and saving

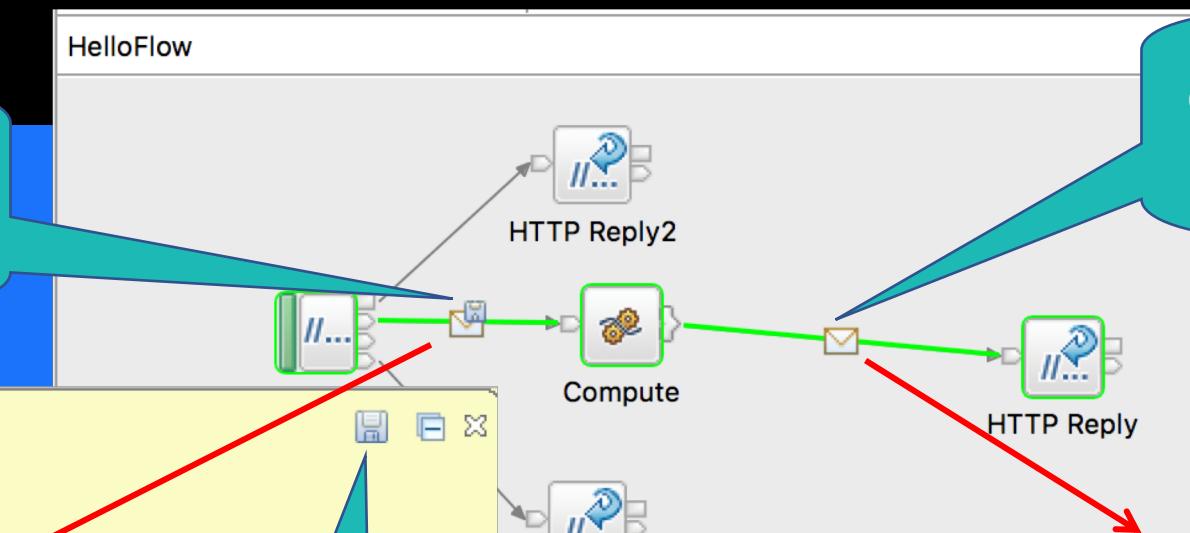
IBM

Click on the connection to see the input message.

Click on the connection to see the output message.

Recorded Message

- ▶ Environment
- ▶ Local Environment
- ▶ Exception List
- ▼ Message
 - ◀message>
 - ▶ Properties
 - ◀/Properties>
 - ▼<HTTPInputHeader>
 - <Content-Type>text/plain; charset=utf-8</Content-Type>
 - <User-Agent>Java/1.8.0_151</User-Agent>
 - <Host>localhost:7800</Host>
 - <Accept>text/html, image/gif, image/jpeg, image/png, */*</Accept>
 - <Connection>keep-alive</Connection>
 - <Content-Length>19</Content-Length>
 - <X-Original-HTTP-Command>POST /</X-Original-HTTP-Command>
 - </HTTPInputHeader>
 - ▼<XMLNSC>
 - <Name>Sanjay</Name>
 - </XMLNSC>
- </message>



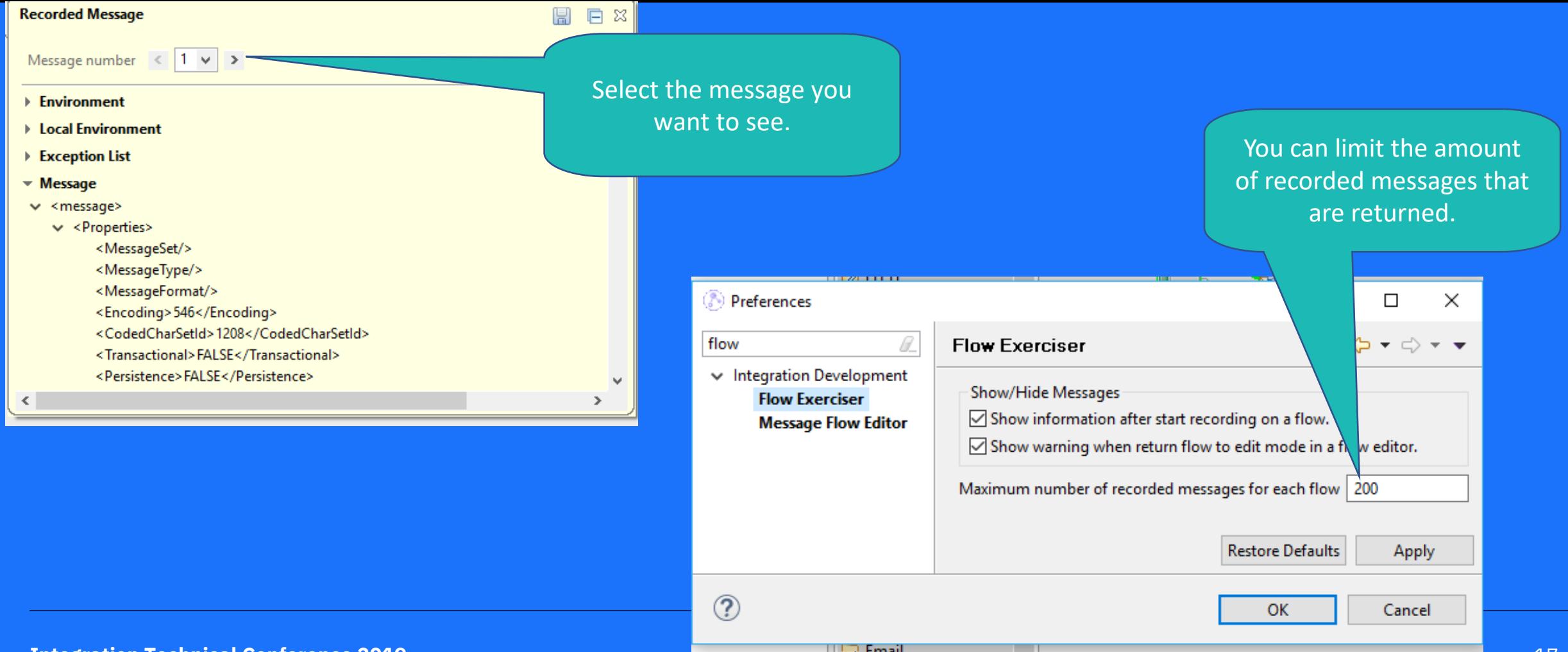
The input message can be saved and used again to inject into the flow.

Recorded Message

- ▶ Environment
- ▶ Local Environment
- ▶ Exception List
- ▼ Message
 - ◀message>
 - ▶ Properties
 - ◀/Properties>
 - ▼<XMLNSC>
 - <Demo>Hello Sanjay</Demo>
 - </XMLNSC>
- </message>

Flow Exerciser – viewing multiple messages

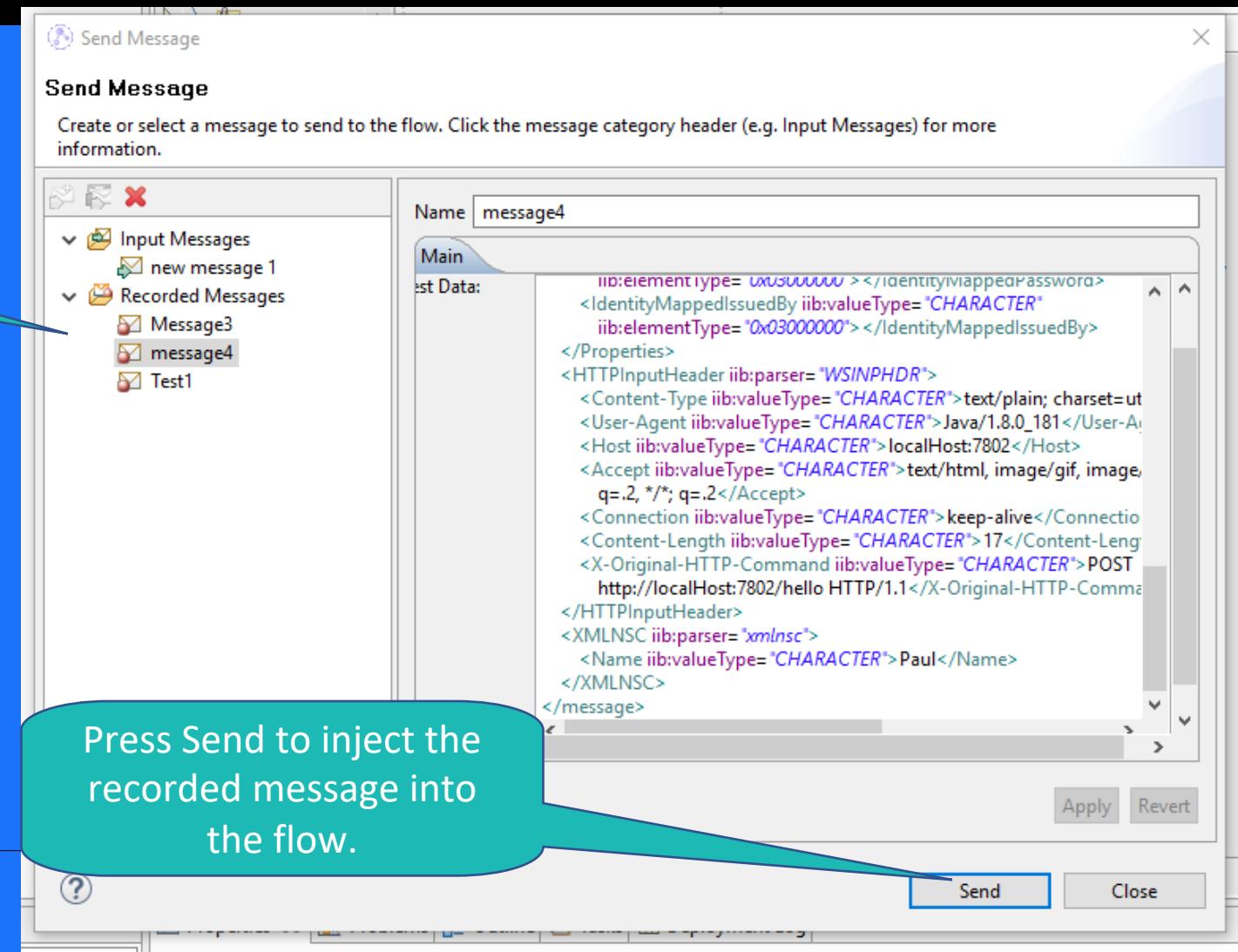
IBM



Flow Exerciser – injecting a recorded message

IBM

Record Messages are displayed here and can be injected into the flow.



Flow Exerciser – using the Admin REST API

IBM

The screenshot shows a web browser window titled "Server APIs - App Connect Ente" with the URL "localhost:7600/apidocs". The page displays the "Start recording" section of the Admin REST API documentation. It includes a "Details" tab, a "Try it" button, and a "POST" method endpoint: `http://localhost:7600/apiv2/applications/{application}/messageflows/{messageflow}/start-recording`. Below the endpoint, the description "Start recording on this messageflow" is visible. The "Parameters" section contains two required parameters: "application" (Application name or id, Example: nenvaojo) and "messageflow" (Messageflow name or id, Example: odasi). A large green callout bubble points to the "Parameters" section with the text: "These APIs are used by the toolkit for recording message and are publicly documented for customers to use." Another green callout bubble points to the "application" parameter with the text: "The Admin REST API allows you to record, save and inject messages."

Start recording

Details Try it

POST http://localhost:7600/apiv2/applications/{application}/messageflows/{messageflow}/start-recording

Start recording on this messageflow

Parameters

Path

application Required Application name or id
Example
nenvaojo

messageflow Required Messageflow name or id
Example
odasi

The Admin REST API allows you to record, save and inject messages.

These APIs are used by the toolkit for recording message and are publicly documented for customers to use.

Flow Exerciser – using the Admin REST API



```
sanjays-mbp:~ sanjay$ curl -X GET http://localhost:7600/apiv2/data/recoded-test-data?depth=6 | python3 -m json.tool
% Total    % Received % Xferd  Average Speed   Time   Time     Total  Spent   Left  Speed
  0          0       0      3100k      0  --:--:--  0:00:00  3187k
{ "count": 6, "recodedTestData": [
  {
    "checkpoint": {
      "messageFlowData": {
        "integrationServer": "MyServer",
        "application": "HelloApp",
        "isDefaultApplication": false,
        "library": "",
        "messageFlow": "HelloFlow",
        "threadId": 2177580,
        "nodes": [
          {
            "propagationType": "terminal",
            "source": {
              "name": "HTTP Input",
              "identifier": "Helloflow#FCMComposite_1_1",
              "type": "ComIbmInputNode",
              "terminal": "out",
              "inputNode": true
            },
            "target": {
              "name": "Compute",
              "identifier": "HelloFlow#FCMComposite_1_3",
              "type": "ComIbmComputeNode",
              "terminal": "in"
            }
          }
        ],
        "sequenceData": {
          "serverSequenceNumber": 1,
          "flowSequenceNumber": 1,
          "threadSequenceNumber": 1,
          "connectionSequenceNumber": 1,
          "timestamp": "2019-04-24 11:52:47.854947"
        },
        "correlationData": {
          "invocationUUID": "79f2eb75-9724-4dea-83be-e8cc40d482b8",
          "inputMessageUUID": "79f2eb75-9724-4dea-83be-e8cc40d482b8"
        }
      }
    }
  }
]
```

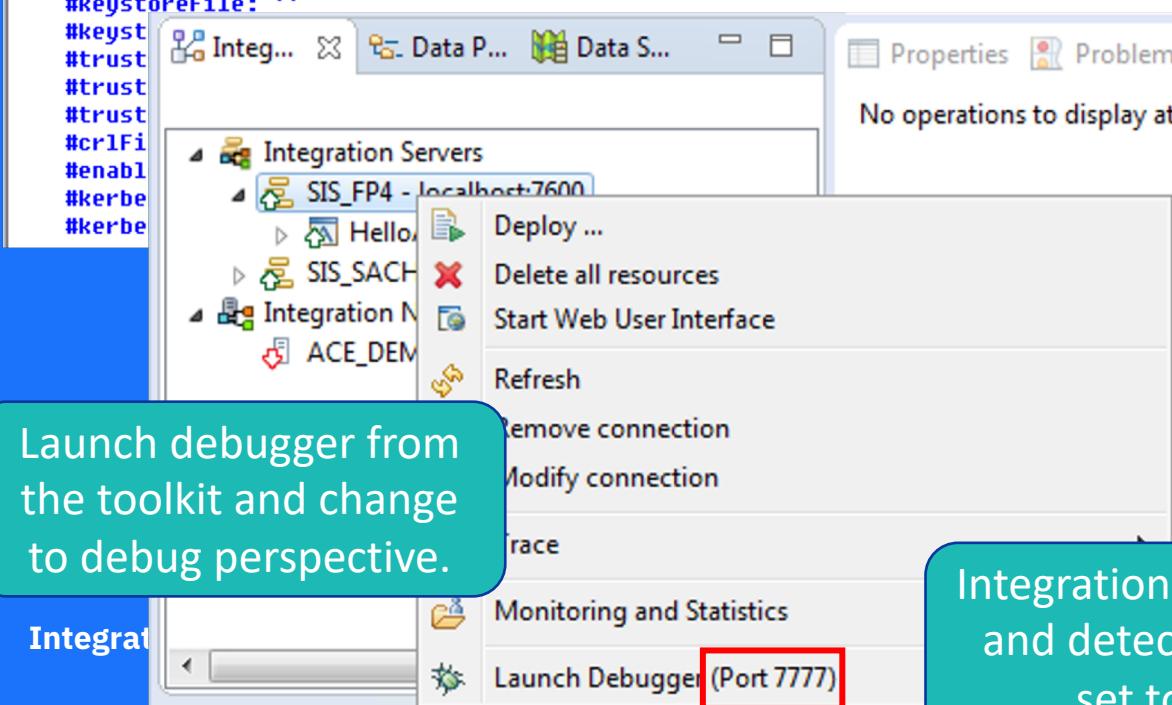
Action	Method	Example URI
Start Recording	POST	http://localhost:7600/apiv2/applications/App1/messageflows/Flow1/start-recording
View and path and save the message.	GET	http://localhost:7600/apiv2/data/recoded-test-data?application=App1&message_flow=Flow1&node_identifier=FCMComposite_1_1
Start Injection	POST	http://localhost:7600/apiv2/applications/App1/messageflows/Flow1/start-injection
Inject Message	POST	http://localhost:7600/apiv2/applications/App1/messageflows/Flow1/nodes/HTTP%20Input/inject Body: { " testData": {...} }
Stop Injection	POST	http://localhost:7600/apiv2/applications/App1/messageflows/Flow1/stop-injection
Stop Recording	POST	http://localhost:7600/apiv2/applications/App1/messageflows/Flow1/start-recording

Flow Debugger

ResourceManagers:

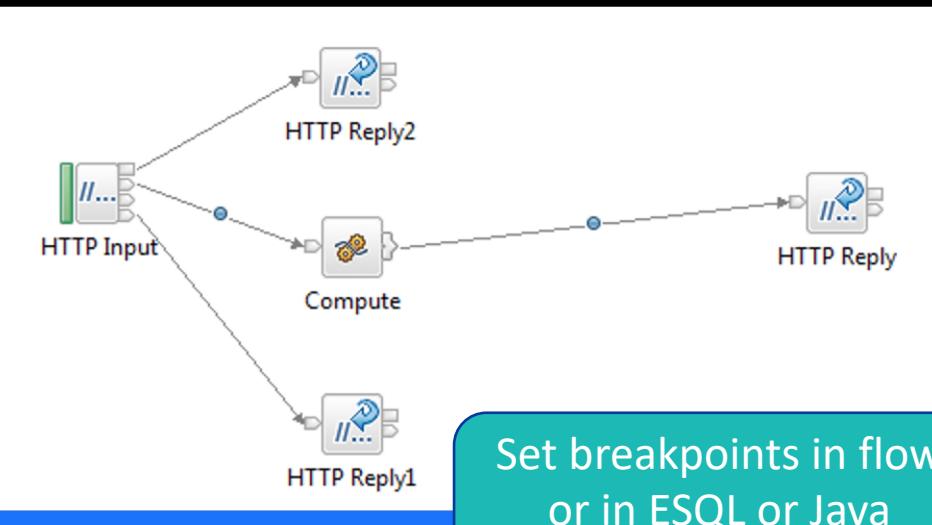
```
JVM:  
  #jvmVerboseOption: 'none'  
  #jvmDisableClassGC: ''  
  #jvmEnableIncGC: ''  
  #jvmShareClasses: ''  
  #jvmNativeStackSize: -1  
  #jvmJavaOSStackSize: -1  
  #jvmMinHeapSize: 33554432      # minimum JVM heap size in bytes (32MB)  
  #jvmMaxHeapSize: 268435455    # maximum JVM heap size in bytes (256MB)  
  jvmDebugPort: 7777            # Set non-zero to activate JVM debug port for Toolkit debugging  
  #jvmSystemProperty: ''  
  #keystoreType: ''  
  #keystoreFile: ''  
  #keyst...  
  #trust...  
  #trust...  
  #trust...  
  #crlFi...  
  #enabl...  
  #kerbe...  
  #kerbe...
```

Set JVM port in server.conf.yaml



Launch debugger from the toolkit and change to debug perspective.

Integration Server starts and detects JVM port set to 7777.



Set breakpoints in flow or in ESQL or Java Compute code.

```
CREATE COMPUTE MODULE HelloFlow_Compute  
CREATE FUNCTION Main() RETURNS BOOLEAN  
BEGIN  
  -- CALL CopyMessageHeaders();  
  -- CALL CopyEntireMessage();  
  
  SET OutputRoot.XMLNSC.Demo = 'Hello ' || InputRoot.XMLNSC.Name || ' !';  
  
  RETURN TRUE;  
END;
```

```
c:\Users\sanjayn\MessageBroker\11.0.0.4>IntegrationServer --name SIS_FP4 --work-dir c:\Users\sanjayn\MessageBroker\11.0.0.4\work_fp4  
....2019-04-24 11:13:59.708736: .2019-04-24 12:13:59.798012: Integration server 'SIS FP4' starting initialization; version '11.0.0.4' (64-bit)  
.....Listening for transport dt_socket at address: 7777  
....2019-04-24 12:14:09.183627: IBM App Connect Enterprise admin security is inactive.  
2019-04-24 12:14:09.343780: The HTTP Listener has started listening on port '7600' for admin http connections.  
2019-04-24 12:14:09.385678: Integration server has finished initialization.
```

Flow Debugger

IBM

The screenshot shows the IBM App Connect Enterprise Toolkit interface during a debugging session. The main window displays a flow diagram with nodes: 'HTTP Input', 'Compute', and 'HTTP Reply1', 'HTTP Reply2'. A yellow circle highlights a breakpoint on the connection between the 'Compute' node and 'HTTP Reply1'. A blue callout bubble points to this circle with the text: "Flow stops at designated breakpoints."

The top menu bar shows "Debug - HelloApp/HelloFlow.msgflow - IBM App Connect Enterprise Toolkit - C:\Users\sanjay\IBM\ACET11\workspace_conference_TechCon_Dallas_2019". The toolbar includes various icons for file operations, navigation, and debugging.

On the left, the "Palette" lists integration technologies: MQPolicy.policyxml, SendMsgToMQ.msgflow, MQPolicy, WebSphere MQ, MQTT, Kafka, JMS, HTTP, REST, Web Services, WebSphere Adapters, and Routing.

The "Variables" view shows the input message structure:

Name	Value
Message	
Properties	
HTTPInputHeader	
XMLNSC	
Name	Sanjay

The "Breakpoints" view shows the current state of the flow.

The "Outline" view shows the flow structure: HelloFlow > HTTP Input > Compute > HTTP Reply.

The bottom "Variables" view shows the output message structure:

Name	Value
OutputRoot	
Properties	
XMLNSC	
Demo	Hello Sanjay !!
OutputLocalEnvironment	
OutputExceptionList	

A blue callout bubble points to the "Variables" view with the text: "Examine input message tree." Another blue callout bubble points to the bottom "Variables" view with the text: "Examine output message tree."

Testing a DFDL schema file



Non XML data (both text and binary) is defined using DFDL schemas

- Data Format Description Language standardised through the Open Grid Forum
- DFDL schemas replace message set projects and the MRM message domain
- Input nodes support DFDL alongside XMLNSC, JSON, MRM etc

The screenshot shows the IBM Integration Designer interface with three main windows:

- simpleCSV.txt**: A text editor window containing a CSV file with five records. The first record is a header row with fields: last, first, middle, address, DOB. Subsequent records provide details for Smith, Robert, Brandon, Johnson, John, Martin, Jones, Arya, Cat, Bonham, John, Henry, and Walters, Amanda, Louise.
- CSV.xsd**: A schema editor window titled "Messages". It displays a hierarchical tree of elements:
 - CSV_message (Element)
 - sequence (1, 1)
 - header (1, 1)
 - sequence (1, 1)
 - head_field1 (string, 1, 1, head_value1)
 - head_field2 (string, 1, 1, head_value2)
 - head_field3 (string, 1, 1, head_value3)
 - head_field4 (string, 1, 1, head_value4)
 - head_field5 (string, 1, 1, head_value5)
 - record (unbounded)
 - sequence (1, 1)
 - field1 (string, 1, 1, value1)
 - field2 (string, 1, 1, value2)
 - field3 (string, 1, 1, value3)
 - field4 (string, 1, 1, value4)
 - field5 (string, 1, 1, value5)
 - Representation Properties**: A properties editor window for the CSV_message element. It shows settings for the message representation, including:
 - Comment**: <default format>
 - General**:
 - Data Format Referrer: <default format>
 - Encoding (code page): <dynamically set>
 - Byte Order: bigEndian
 - Ignore Case: no
 - Fill Byte: 0
 - Content**:
 - Length Kind: delimited
 - Nillable: false
 - Occurrences**:
 - Min Occurs: 1
 - Max Occurs: 1
 - Alignment**
 - Delimiters**

Example CSV message modelled by a DFDL schema

When you have defined your DFDL schemas, you can test them in the editor before using a deployed message flow.

Testing a DFDL schema file

IBM

The screenshot shows the IBM DFDL Test tool interface. On the left, the DFDL Editor displays a schema named CSV.xsd with a tree view of elements like CSV_message, sequence, header, and various fields. A red box highlights the 'DFDL Test - Parse' tab in the toolbar. The central area shows a message titled 'CSV_message' with a table of parsed data. A green callout bubble points to this table with the text: 'Generate a logical instance document from your DFDL schema'. The bottom section shows the 'Parsed Input' window with sample CSV data and a green callout bubble pointing to it with the text: 'Test-parse sample data against your DFDL schema'. The status bar at the bottom indicates 'Parsing completed successfully.'

CSV.xsd

Test Parse Model Test Serialize Model Show properties Show all sections Focus on selected Show quick outline Create logical instance

Messages

A message is a global element that models an entire document of data.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
CSV_message					
sequence		1	1		
header		1	1		
sequence		1	1		
head_field1	string	1	1	head_value1	
head_field2	string	1	1	head_value2	
head_field3	string	1	1	head_value3	
head_field4	string	1	1	head_value4	
head_field5	string	1	1	head_value5	

Navigator Problems DFDL Test - Parse DFDL Test - Serialize DFDL Test - Trace

DFDL Test - Parse: Runs the DFDL parser with the selected physical input data and selected message, Status: Parsing completed: Wed Apr 24 14:29:07 BST 2019

Input Data: /CommaSeparatedValues/simpleCSV.txt

Parsed Input

Characters

```
1 last,first,middle,address,DOB
2 Smith,Robert,Brandon,"Boleskin House, Loch Ness",1988-03
3 Johnson,John,Martin,"7, Topcliffe Drive, Farnborough",19
4 Jones,Arya,Cat,"274, High St, Princes Risborough",1986-0
5 Bonham,John,Henry,"The Old Hyde, Redditch",1948-05-31
6 Walters,Amanda,Louise,"1 The Woodlands, Saunders Lane, Romsey",1966-08-01
7
```

Selection in DFDL Editor
Selected: CSV_message : <Anonymous> (complex) Repeating index: 1 Range in parsed input: 0 - 350 Character Selection In Input Row: 0 Column: 0 Byte Selection In Input Offset: 0 Length: 0

DFDL Test - Logical Instance

Data source: <From 'DFDL Test - Parse' view>

Message: CSV_message (/Users/sanjayn/IBM/ACET11/workspace_conference_TechCon_Dallas_2)

Tree View XML View

Name	Type	Value
header		
head_field1	xs:string	last
head_field2	xs:string	first
head_field3	xs:string	middle
head_field4	xs:string	address
head_field5	xs:string	DOB
record		
field1	xs:string	Smith
field2	xs:string	Robert
field3	xs:string	Brandon
field4	xs:string	Boleskin House...
field5	xs:string	1988-03-24

Tips:

- Selecting an element in the DFDL editor will cause the parsed input to focus only on data pertaining to the selected element.
- The view menu on the view toolbar provides options to control how the data is displayed in the view. Click the arrow icon on the toolbar or [here](#) to open the menu.
- To view the logical instance that was created by the DFDL parser, click the Open DFDL Logical Instance View toolbar button, or click [here](#).
- To view the trace captured while running the DFDL parser, click the Open DFDL Trace View toolbar button, or click [here](#).

Do not display this message again

Generate a logical instance document from your DFDL schema

Test-parse sample data against your DFDL schema

Testing a DFDL schema file

IBM

The screenshot shows the Rational Application Developer interface with two main windows:

- CSV.xsd View:** This window displays the XML schema definition (CSV.xsd) for a message named "CSV_message". The schema includes a sequence for "header" containing five string fields: "head_field1" through "head_field5", each with a default value ("head_value1" through "head_value5").
- DFDL Test - Logical Instance View:** This window shows a logical instance of the schema. It contains two sections: "header" and "record". The "header" section has five fields: "head_field1" (value "last"), "head_field2" (value "first"), "head_field3" (value "middle"), "head_field4" (value "address"), and "head_field5" (value "DOB"). The "record" section has five fields: "field1" (value "Smith"), "field2" (value "Robert"), "field3" (value "Brandon"), "field4" (value "Boleskin House..."), and "field5" (value "1988-03-24").

A green callout bubble points from the "header" section of the logical instance view towards the text "Test-serialize from a logical instance using data from your DFDL schema".

DFDL Test - Serialize View: This view shows the status of the serialization process. It says "Serialization completed successfully." and provides tips for using the view toolbar and trace capture.

Serialized Output: This section shows the resulting CSV output:

```
1 last,first,middle,address,DOB
2 Smith,Robert,Brandon,"Boleskin House, Loch Ness",1988-03-24
3 Johnson,John,Martin,"7, Topcliffe Drive, Farnborough",1986-01-23
4 Jones,Arya,Cat,"274, High St, Princes Risborough",1986-02-19
5 Bonham,John,Henry,"The Old Hyde, Redditch",1948-05-31
6 Walters,Amanda,Louise,"1 The Woodlands, Saunders Lane, Romsey",1966-08-01
7
```

At the bottom, there are "Character Selection In Output" and "Byte Selection In Output" buttons, along with "Row: 0 | Column: 0" and "Offset: 0 | Length: 0" fields.

Demo Time!!

Performance Considerations

Why should you care about performance?



- Optimising your performance can lead to real \$\$ savings... Time is Money! If you design your solution well....
 - You could increase your message throughput
 - You could reduce processing costs to achieve the same throughput
- Understand your Service Level Agreement (SLA):
 - What are the requirements for Quality of Service: transactional processing, assured delivery, availability, restart/recovery.
- Attaining high performance is dependent of the nature of your application!
 - What is the size of the message you are processing?
 - What transport are you using?
- It's better to design your message flows to be as efficient as possible from day one
 - Follow best practices so your flows are as efficient as possible.
 - Solving bad design later on costs much more. Working around bad design can make the problem worse!

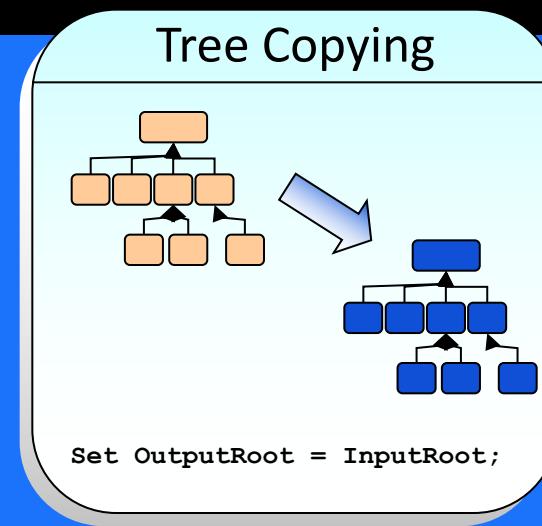
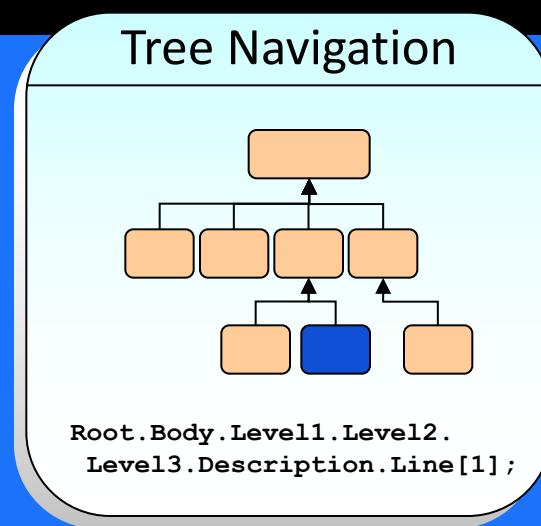
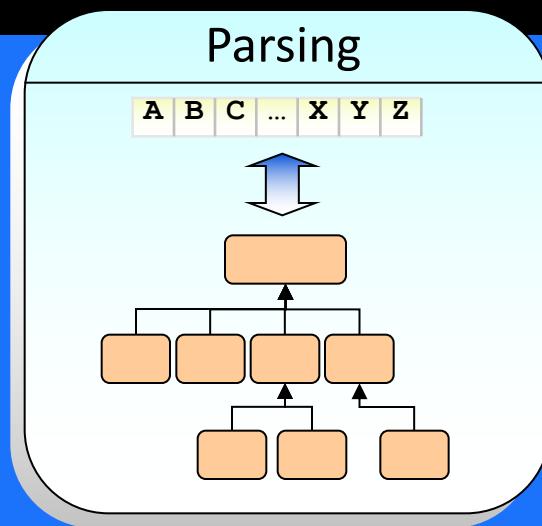


Design, Code, Run, Measure...Change, Run, Measure



- Delivering the right level of performance straight away is unlikely (well done if you do it!) Start with a single instance if the flow on the server
- Performance evaluation and improvement is an iterative process:
 - Design
 - Code
 - Run
 - Measure
 - Identify hotspotsTest flows in isolationTake a baseline before changes are madeMake ONE change at a time and re-test to measure impactRun a driver application separate to your ACE machineUse real data and avoid pre-loading queues.
- Important to have the philosophy, process, infrastructure and tools to enable this

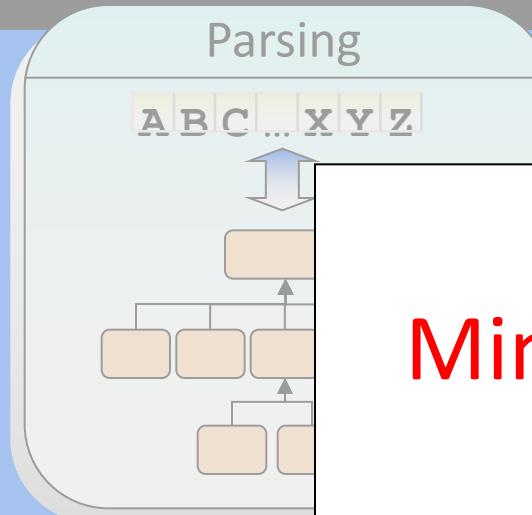
What are the main performance costs in message flows?



Techniques to Help Optimise Performance

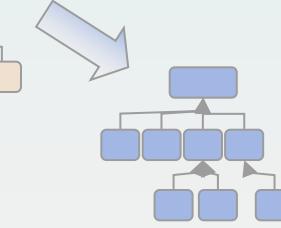


Minimise the number of times each is used and the cost each time



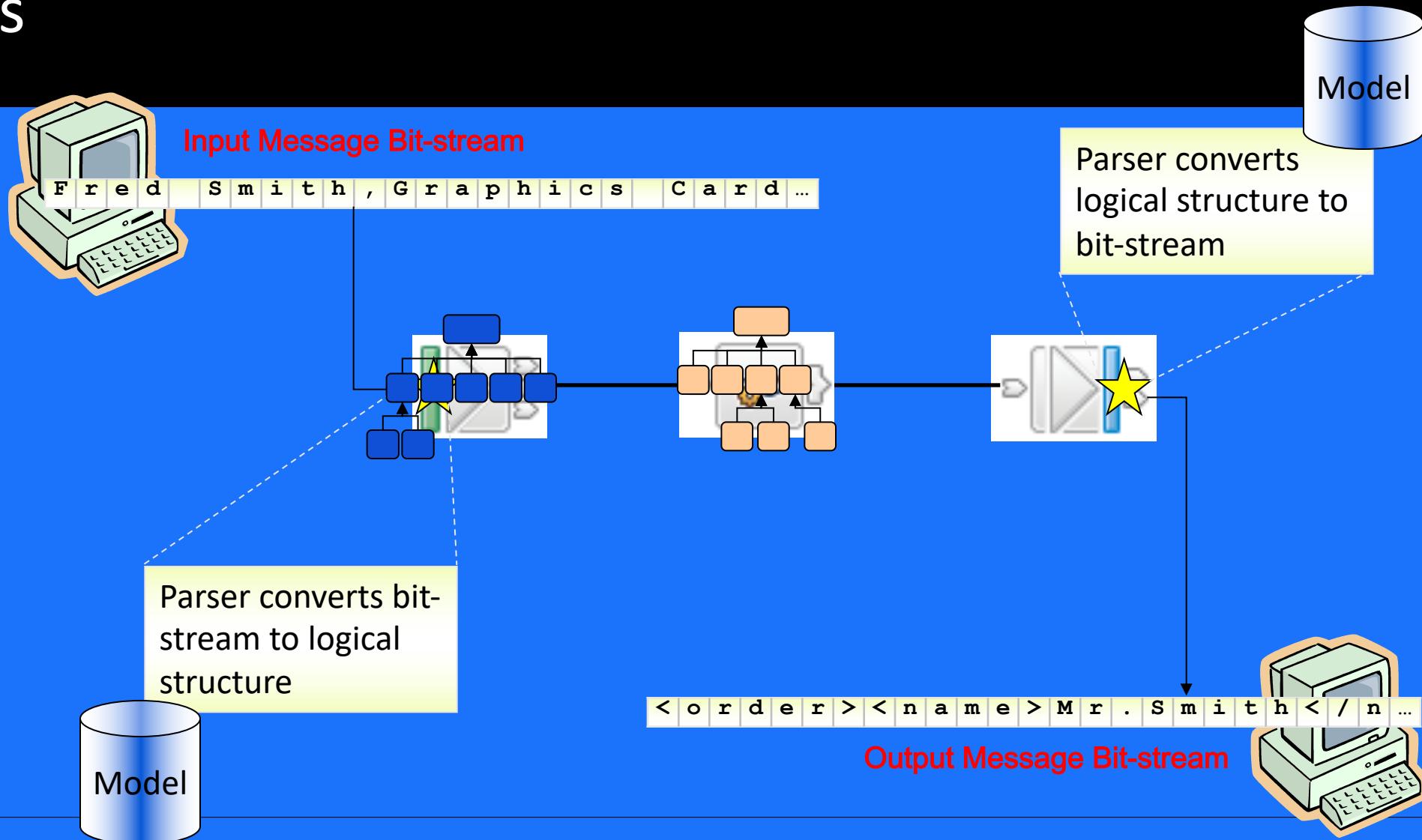
Tree Navigation

Tree Copying



`tRoot = InputRoot;`

Parsers



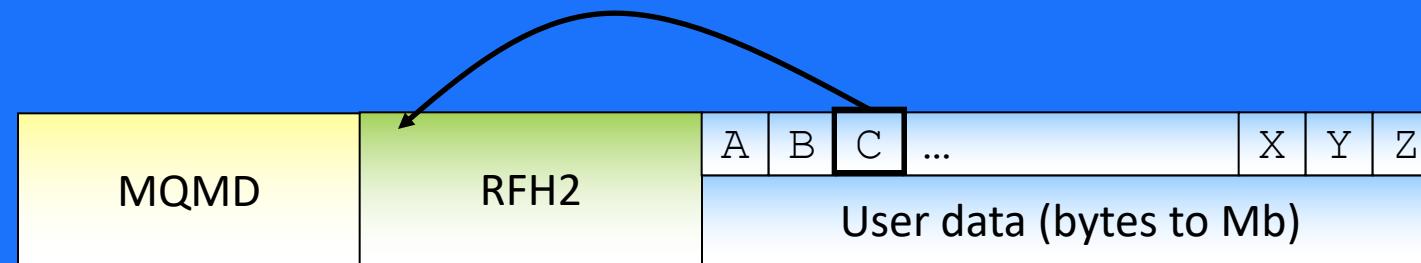
Parsing

- The means of populating and serializing the tree
 - Multiple Parsers available: XMLNSC, JSON, DFDL, MIME, JMSMap, JMSStream, BLOB, SOAP
 - A Parser may use a model:
 - Parse a message that is not self-defining (CSV, COBOL, C messages)
 - Validate a message
 - Used for developing transformation logic (e.g. graphical mapper)
- Cost of parsing:
 - Message structure (more complicated the message structure, the more it will cost to parse!!)
 - Type of message (cheaper to parse a fixed length message than a variable length message)
- Several ways of minimizing parsing costs
 - Use cheapest parser possible, e.g. XMLNSC for XML parsing
 - Identify the message type quickly
 - Use Parsing strategies
 - Parsing avoidance
 - Partial parsing
 - Opaque parsing

Parser avoidance



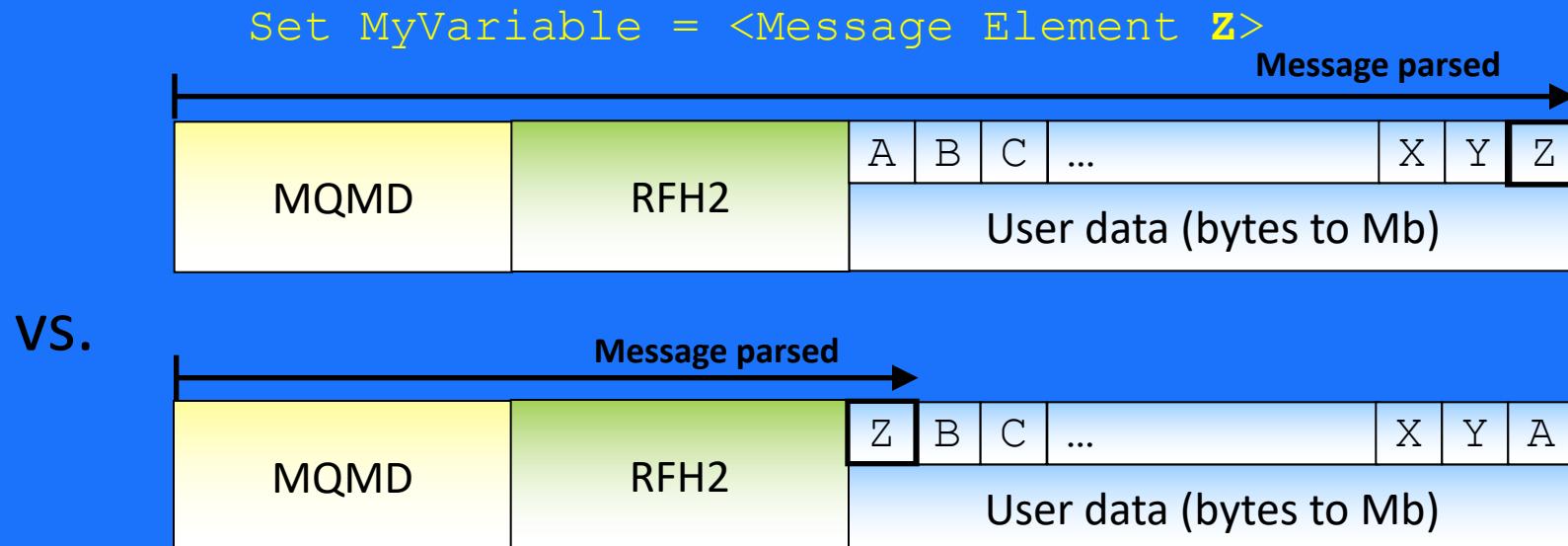
- If possible, avoid the need to parse at all!
 - Consider only sending changed data
 - Promote/copy key data structures to MQMD, MQRFH2 or JMS Properties
 - May save having to parse the user data
 - Particularly useful for message routing



Partial parsing



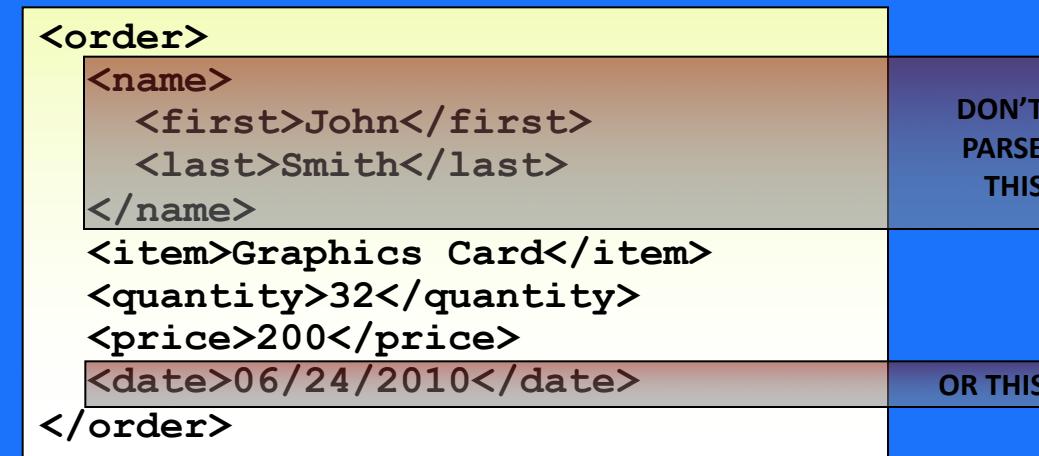
- Typically, the parser parses elements up to and including the required field
 - Elements that are already parsed are not reparsed
- If possible, put important elements nearer the front of the user data



Typical Ratio of CPU Costs	1K msg	16K msg	256K msg
Filter First	1	1	1
Filter Last	1.4	3.4	5.6

Opaque parsing

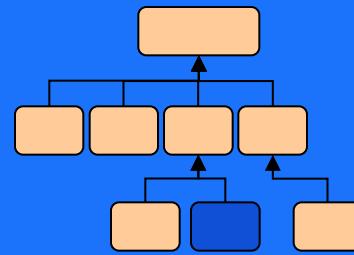
- Treat elements of an XML document as an unparsed BLOB
- Reduces message tree size and parsing costs
- Cannot reference the sub tree in message flow processing
- Configured on input nodes (“Parser Options” tab)



Navigation



- The logical tree is walked every time it is evaluated
 - This is not the same as parsing!

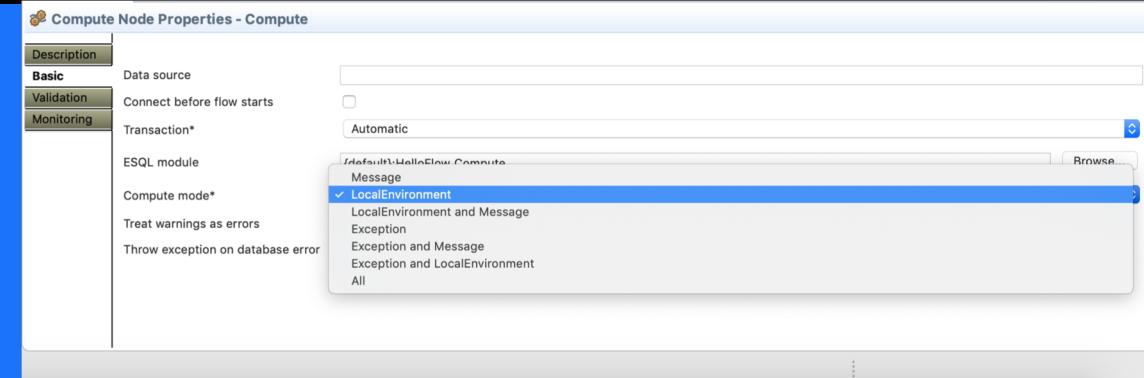
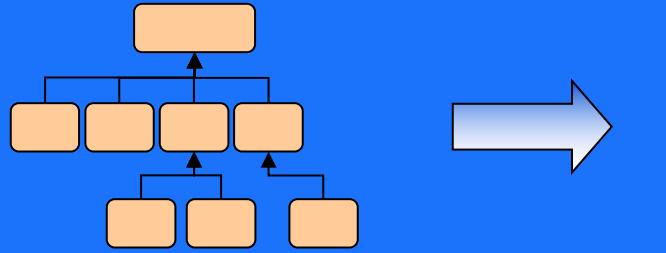


```
SET Description =  
Root.Body.Level1.Level2.Level3.Description.Line[1];
```

- Long paths are inefficient
 - Minimise their usage, particularly in loops
 - Use reference variables/pointers (ESQL/Java)
 - Build a smaller message tree if possible
 - Use compact parsers (e.g. XMLNSC)
 - Use opaque parsing

Message tree copying

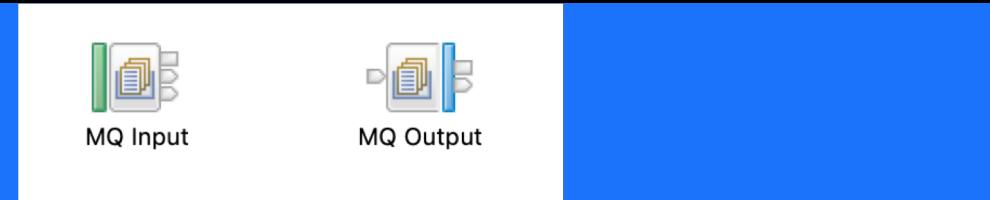
```
SET OutputRoot.XMLNSC.A = InputRoot.XMLNSC.A;
```



- Message tree copying causes the logical tree to be duplicated in memory... and this is computationally expensive
- Reduce the number of times the tree is copied
 - Reduce the number of *Compute* and *JavaCompute* nodes in a message flow
 - See if “Compute mode” can be set to not include “message”
 - Copy at an appropriate level in the tree (copy once rather than for multiple branch nodes)
 - Copy data to the Environment (although changes are not backed out)

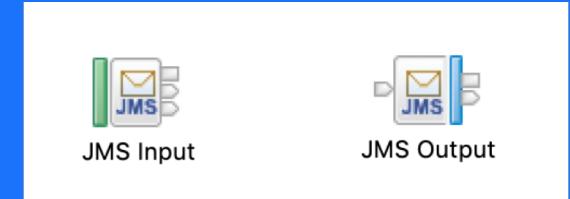
MQ

- Tune the QM and associated QM's, logs, buffer sizes
- Avoid unnecessary use of persistent messages,
- But use persistent messages as part of a co-ordinated transaction
- Use fast storage if using persistent messages



JMS

- Follow MQ tuning if using MQ JMS
- Follow provider instructions



SOAP/HTTP

- Use persistent HTTP connections
- Use embedded execution group-level listener for HTTP nodes



File

- Use cheapest delimiting option (see reports)
- Use fast storage



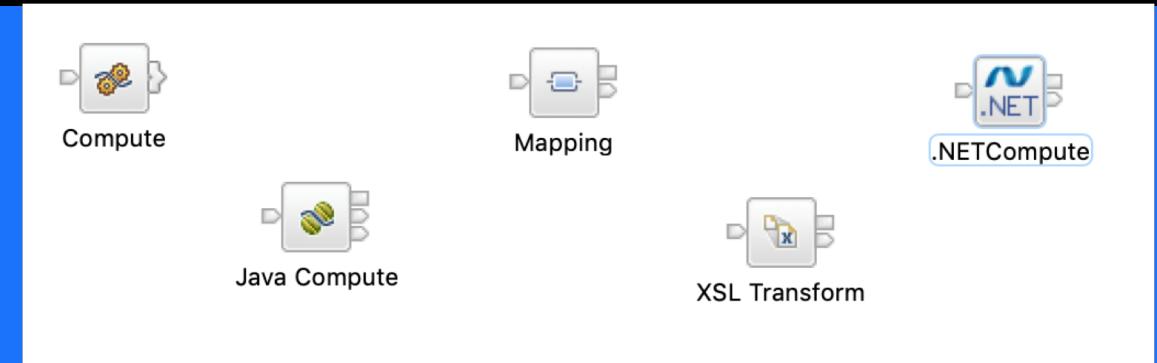
Transformation Options

Transformation Options



- App Connect Enterprise has several transformation options:

- Mapping
- XSLT
- ESQL
- Java
- .NET



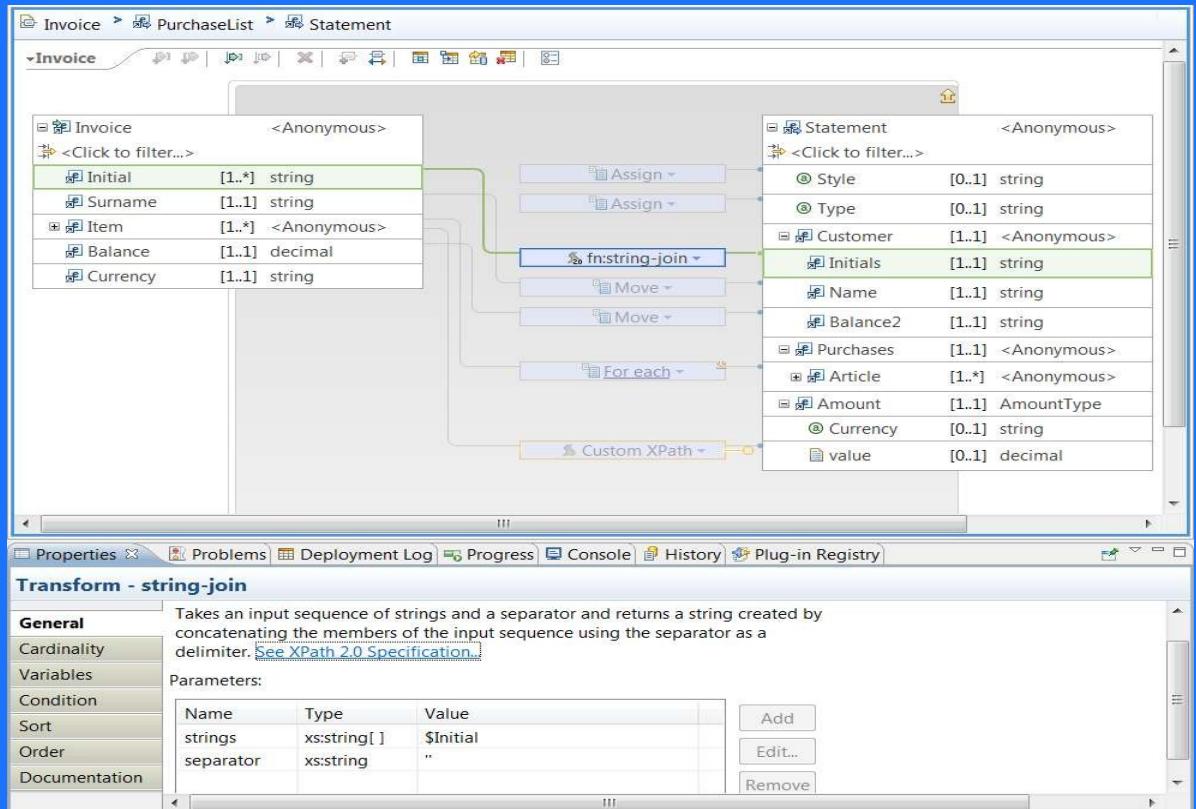
- Reflects the importance of transformation in connectivity solutions
- Every transformation option has strengths and weaknesses!
 - Performance and scalability
 - Backend integration
 - Skill sets and learning curve
 - Developer usability
 - Portability and maintenance
- Use a transformation technology appropriate to the problem at hand!

Mapping Node

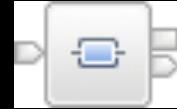


IBM

- Message flows are a kind of ‘graphical programming’
 - Wires define message routing
- Maps extend this to message transformation
 - Lines drawn between structural elements of the message
- Simple to use – drag and drop transformation
- Highly configurable using XPath 2.0
 - Versatile expression language – W3C standard
 - Large function library
- Ability to call user-defined functions/methods
 - ESQL and Java (static methods)
- Can map transport headers and **LocalEnvironment**
 - Not **Environment** or **ExceptionList**
- Can generate multiple output messages
 - Split a document into different structures
 - Shred a large document with repeating elements
- Database support
 - Select, Insert, Update, Delete, Stored Procedures



Mapping Node – at a glance



IBM

Functionality	Ease of Use	Performance
<ul style="list-style-type: none">▪ Multi domain support▪ Integrates with RDBMS▪ Rich function library<ul style="list-style-type: none">– Extensible with Java & ESQL	<ul style="list-style-type: none">▪ Gentle learning curve▪ Graphical drag and drop▪ Minimal ‘programming’ skills	<ul style="list-style-type: none">▪ Compiles and runs in dedicated engine▪ Benefits from JIT technology
Standards	Maintenance	Portability
<ul style="list-style-type: none">▪ W3C XPath 2.0 expression language▪ Structures modelled in XML Schema (XSD)	<ul style="list-style-type: none">▪ Visual representation assists understanding	<ul style="list-style-type: none">▪ Graphical Data Mapper used across many IBM products<ul style="list-style-type: none">– Including DataPower MDM Server, RAD

Compute Node



IBM

- Excellent for database interaction
- Syntactically similar to SQL
- Invoke static Java methods and database stored procedures
- Supports declarative and procedural programming styles
- Powerful SELECT statement can be applied to messages as well as database tables (or a mix of the two at the same time – can even be nested!)
- Access to all message domains
- Can address all message headers and environment trees
- Toolkit support with editor for syntax highlighting and context assist
- ESQL debugger integrated with Flow Debugger

- Field References
 - Path syntax to address the tree elements
 - Starts with ‘correlation name’ to identify root of tree

```
SET OutputRoot = InputRoot; -- copies the whole message
```

```
SET OutputRoot.MQMD = NULL; -- removes the MQMD header
```

```
SET OutputRoot.XMLNSC.doc.title = -- will generate the output  
InputBody.session[4].title; -- tree if it doesn't exist
```

```
SET OutputRoot.XMLNSC.Library.Publication[] =  
InputBody.library.books.book[] -- copies all elements  
(deep copy)
```

ESQ performance statistics



Enable ESQL performance statistics reporting by setting properties in server.conf.yaml

```
ESQL:  
  #castWithAccurateZoneOffsets: false      # Set ESQL to use the time zones that are stored with Time, Timestamp, and Date data types when you cast to and from GMTime and GMTimestamp data types.  
  #alwaysCastToLocal: true                 # Set ESQL to convert the specified time into the local time zone of the integration node/server when a String is cast to a Time or TimeStamp with a Format.  
  #useICUStringManipulation: false        # Set ESQL to provide full support for UTF-16 surrogate pairs.  
  #allowPropagateWithSharedLockHeld: false  # Set ESQL to propagate the message even if a lock is held on a shared variable.  
  performanceStatsReportingOn: true         # Set ESQL to capture performance statistics  
  performanceStatsOutputPath: '/Users/sanjayn/ESQL_perf_stats'    # Sets the path that ESQL performance statistics are captured to
```

File written to /Users/sanjayn/ESQL_perf_stats/integration_server_MyServer_ESQLPerfStats.txt

GMT Timestamp	TID	[uSec]	[Cycles]	[Address]	RDL:
2019-04-25 07:59:39.666586	2421964	[2506]	[2476]	[0x00007fe5a4950400]	Statements...
2019-04-25 07:59:39.667714	2421964	[1369]	[1370]	[0x00007fe5a2e17000]	Begin Compute Node HelloFlow.Compute
2019-04-25 07:59:39.667727	2421964	[1349]	[1350]	[0x00007fe5a2f180a0]	SCHEMA "" PATH ""
2019-04-25 07:59:39.667740	2421964	[1327]	[1328]	[0x00007fe5a2f18440]	MODULE HelloFlow_Compute
2019-04-25 07:59:39.667951	2421964	[1108]	[1108]	[0x00007fe5a2f187a0]	FUNCTION Main(...)
2019-04-25 07:59:39.668296	2421964	[729]	[731]	[0x00007fe5a2f188e0]	BEGIN
2019-04-25 07:59:39.669043	2421964	[7]	[7]	[0x00007fe5a2e13ff0]	SET OutputRoot.XMLNSC.Demo = 'Hello ' InputRoot.XMLNSC.Name ' !';
2019-04-25 07:59:39.669059	2421964	[1108]	[1108]	[0x00007fe5a2f187a0]	RETURN TRUE;
2019-04-25 07:59:39.669067	2421964	[1327]	[1328]	[0x00007fe5a2f18440]	END;
2019-04-25 07:59:39.669076	2421964	[1349]	[1350]	[0x00007fe5a2f180a0]	END Function Main;
2019-04-25 07:59:39.669083	2421964	[1369]	[1370]	[0x00007fe5a2e17000]	END MODULE HelloFlow_Compute;
2019-04-25 07:59:39.669092	2421964	[2506]	[2476]	[0x00007fe5a4950400]	End Schema "";
					End Compute Node HelloFlow.Compute

Compute Node – at a glance



IBM

Functionality	Ease of Use	Performance
<ul style="list-style-type: none">▪ Full domain support▪ Integrates with RDBMS▪ Rich function library<ul style="list-style-type: none">– Extensible with Java, .NET	<ul style="list-style-type: none">▪ Compact DSL (Domain Specific Language)▪ Path navigation build directly into syntax	<ul style="list-style-type: none">▪ Interpreter closely coupled to message tree<ul style="list-style-type: none">– Excellent tree performance▪ Easy to write poorly performing code though
Standards	Maintenance	Portability
<ul style="list-style-type: none">▪ Built on SQL standard▪ Proprietary extensions	<ul style="list-style-type: none">▪ Built in source code debugger▪ Supports development of reusable code libraries▪ Deployed as source	<ul style="list-style-type: none">▪ Runs on all platforms▪ Unlikely to run anywhere else

JavaCompute



IBM

- General purpose programmable node
- Java 8 (IBM Integration Bus v10, App Connect Enterprise v11)
- API to work with messages and interact with broker
- JAXB support for creating portable transformation logic
- Full XPath 1.0 support for message navigation
- Two general purpose output terminals for message routing
- Supports all message domains, headers and environment messages
- Full IDE support of Eclipse Java Development Tools (JDT)
 - Java debugger integrated with Flow Debugger
- Wizard driven
 - Skeleton Java code is created
- User codes logic in **evaluate()**method

- The incoming message is passed into **evaluate()**as part of a message assembly
 - MbMessageAssembly encapsulates four MbMessage objects
 - Message - the incoming message
 - Local environment - local scratch pad work area
 - Global environment - global scratch pad work area
 - Exception list - the current exception list
- Message assembly is propagated to an output terminal

A screenshot of an Eclipse Java Development Tools (JDT) code editor window. The title bar says "JDBCTestFlow_JavaCompute.java". The code is as follows:

```
package test.jdbc;

import java.sql.Connection;

public class JDBCTestFlow_JavaCompute extends MbJavaComputeNode {

    public void evaluate(MbMessageAssembly inAssembly) throws MbException {
        MbOutputTerminal out = getOutputTerminal("out");

        MbMessage inMessage = inAssembly.getMessage();
        MbMessageAssembly outAssembly = null;

        PreparedStatement selectStmt = null;

        try {
            // create new message as a copy of the input
            MbMessage outMessage = new MbMessage(inMessage);
            outAssembly = new MbMessageAssembly(inAssembly, outMessage);
            // -----
            // Add user code below

            Connection conn = getJDBCType4Connection("[MyPolicies]:DB2Policy",
                JDBC_TransactionType.MB_TRANSACTION_AUTO);

            String stmt = "SELECT * FROM SANJAYN.CUSTOMERS";
        }
    }
}
```

JavaCompute - at a glance

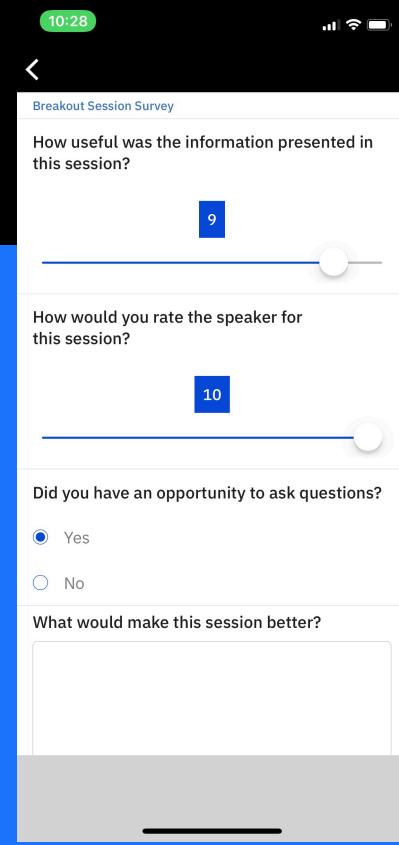
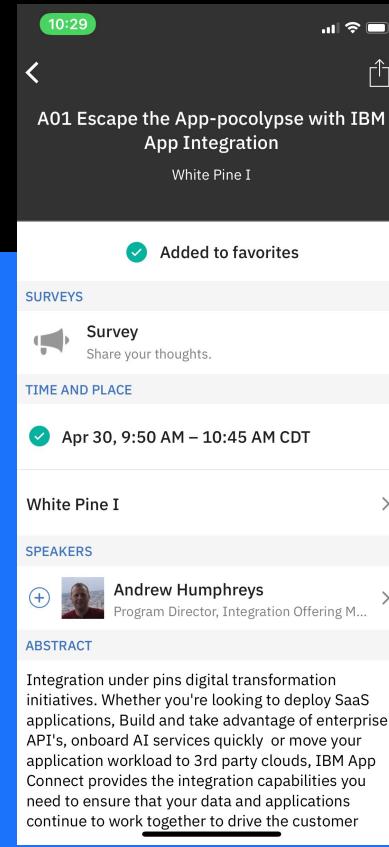
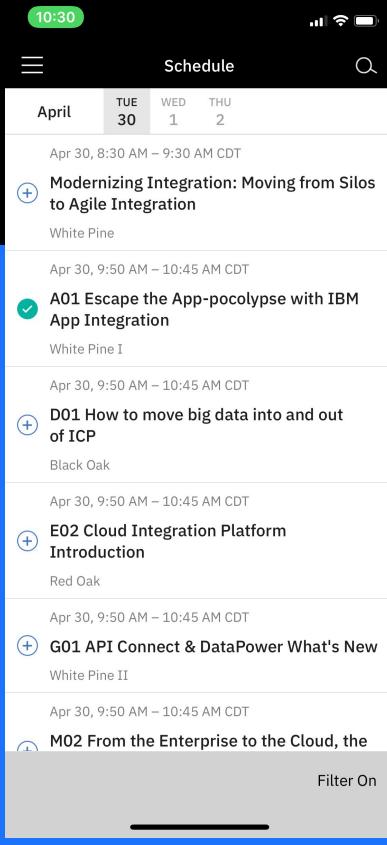


IBM

Functionality	Ease of Use	Performance
<ul style="list-style-type: none">▪ Full J2SE JVM▪ Huge function library<ul style="list-style-type: none">– JDK 7.0– 3rd party JARs▪ Full domain support▪ Excellent I/O capabilities	<ul style="list-style-type: none">▪ Standard Java + IIB API▪ Navigation using XPath▪ API-less transformation using standard JAXB▪ Full JDT tools▪ Beware of multi-threads	<ul style="list-style-type: none">▪ Advanced JIT technology▪ Thin IIB API around native IIB internals▪ Easy to write poorly performing code though
Standards	Maintenance	Portability
<ul style="list-style-type: none">▪ Java<ul style="list-style-type: none">– Ubiquitous– JAXB, JDBC XA, ...▪ W3C<ul style="list-style-type: none">– XPath 1.0	<ul style="list-style-type: none">▪ Built in source code debugger▪ Productivity tools<ul style="list-style-type: none">– JUnit, JavaDoc, ...	<ul style="list-style-type: none">▪ Runs on all platforms▪ JAXB transformation will run on other products and vendors

Summary

- Multiple areas in which recommendations can be applied
 - Design
 - Coding
 - Test
 - Deployment
- Most resilient and best performing systems are those that are:
 - Loosely coupled and have parallel execution (threads and processes)
 - Implications for your message flow design will depend on the systems that IIB interacts with
- Vital to conduct performance testing before production in production like environment
 - Gives time to evaluate and refactor code if needed



IBM

Don't forget to fill out the survey!

Select your session, select survey, rate the session and submit!

Thank You

