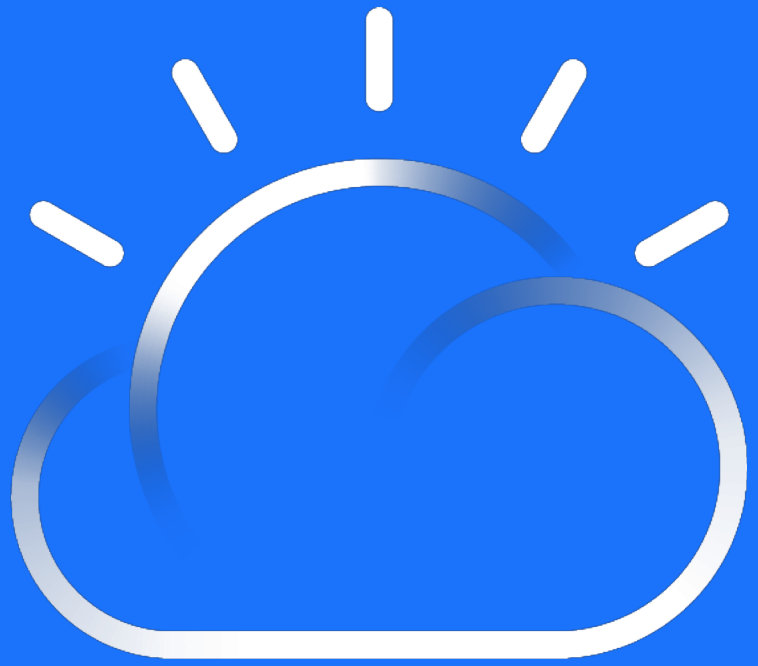
Integration Technical Conference 2019

A07 Application Integration

A07 Becoming an Integration ACE with ACEv11



Sanjay Nagchowdhury
sanjay_nagchowdhury@uk.ibm.com

IBM **Cloud**

IBM

Agenda

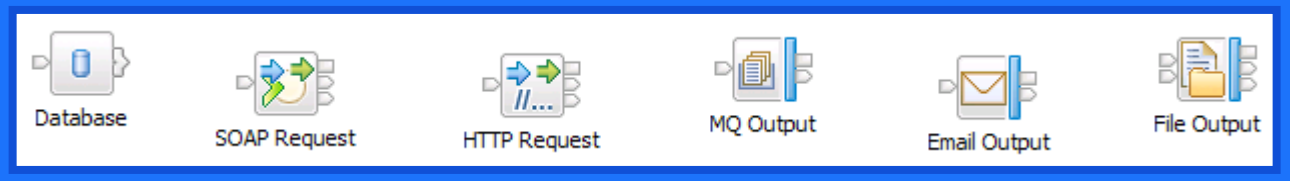
- Message Flow Design
- Test Considerations
- Demo Time!!
- Performance Considerations
- Transformation Options

Message Flow Design

Your connectivity solution

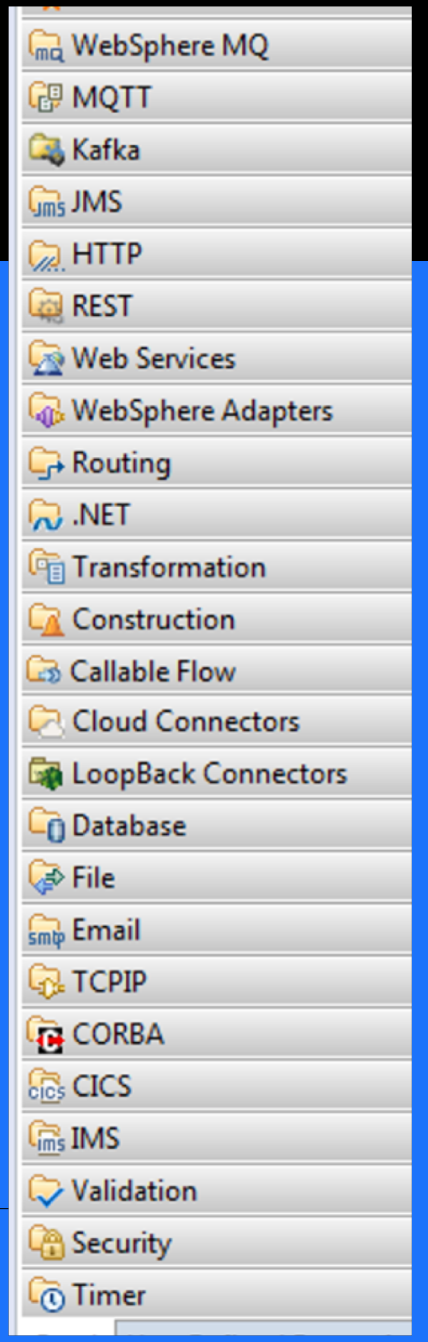
Built-in nodes encapsulate transports, technologies and applications

- 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:

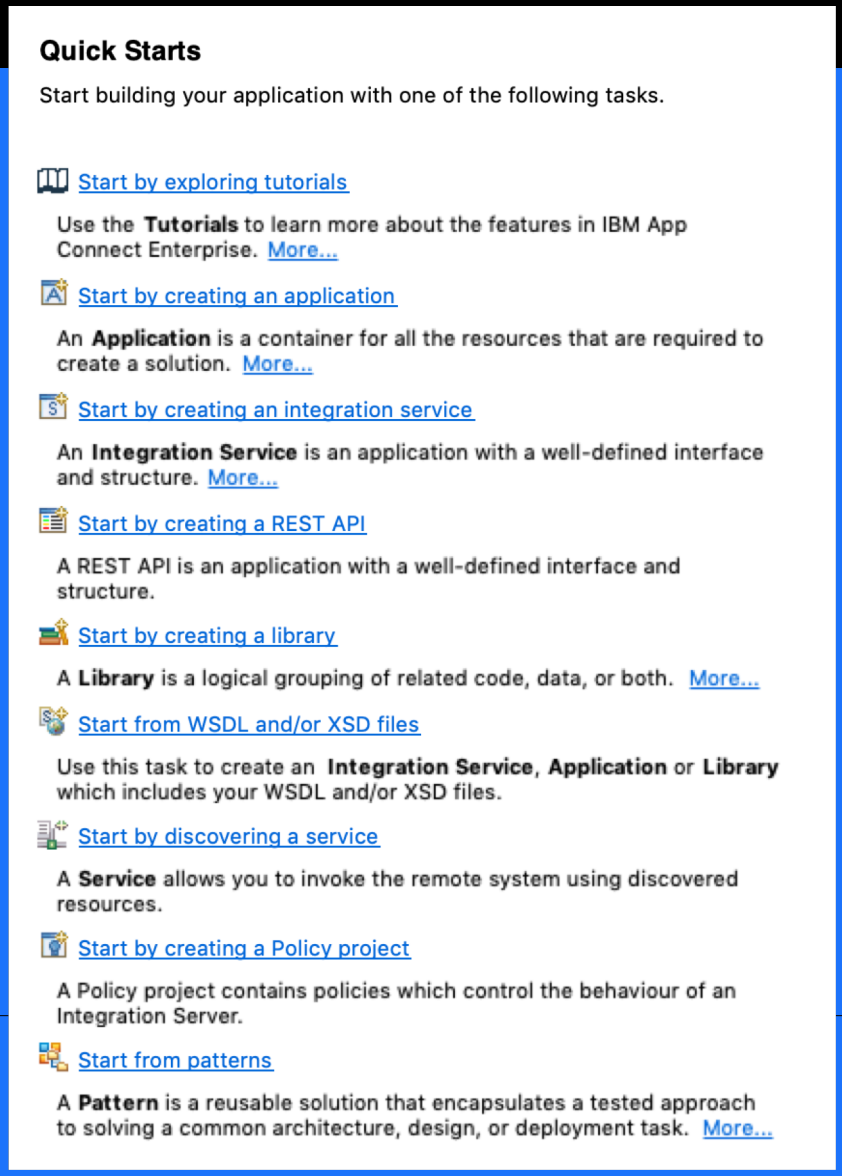
- Connecting with client applications
- Routing messages
- Transforming and enriching messages
- Processing events
- Handling errors in message flows



Developing an integration solution

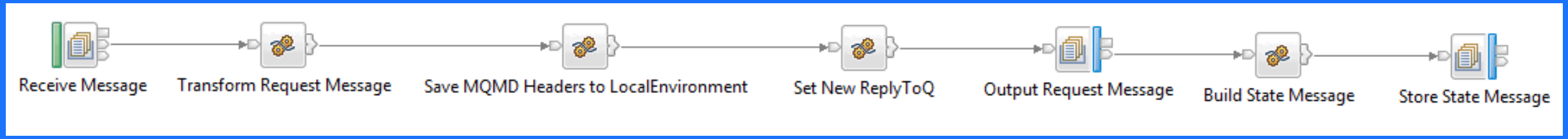
IBM App Connect Enterprise provides different ways of creating your integration solutions.

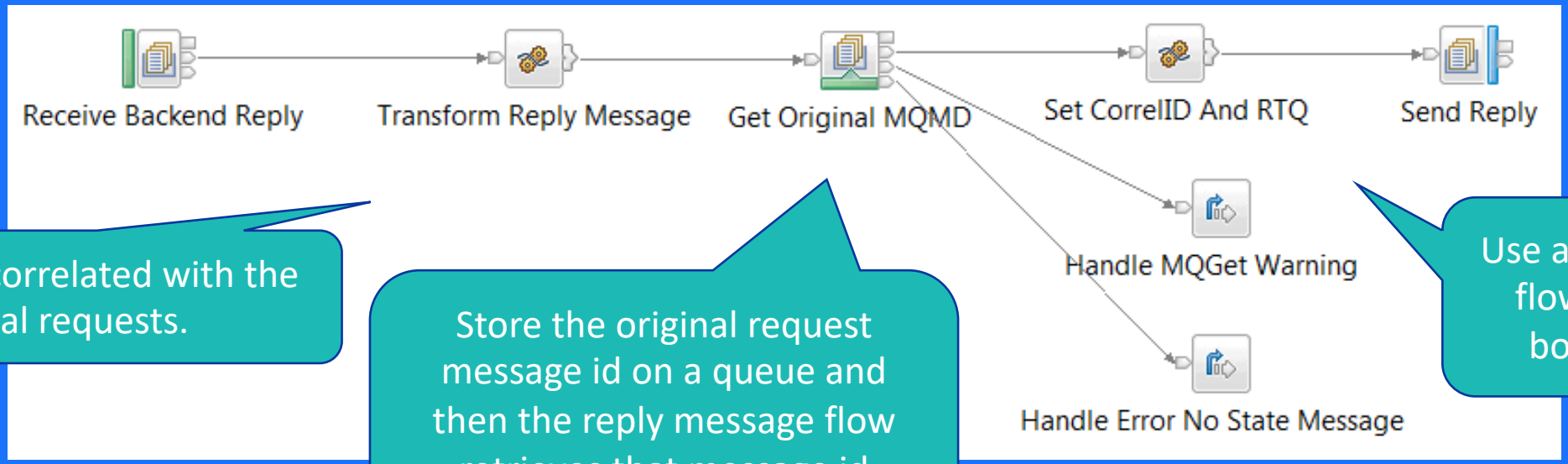
- Using Applications and Libraries
- Using an Integration Service
- Using a REST API
- Using a Policy
- Using Patterns



Message Flow Structures

Asynchronous request-response message





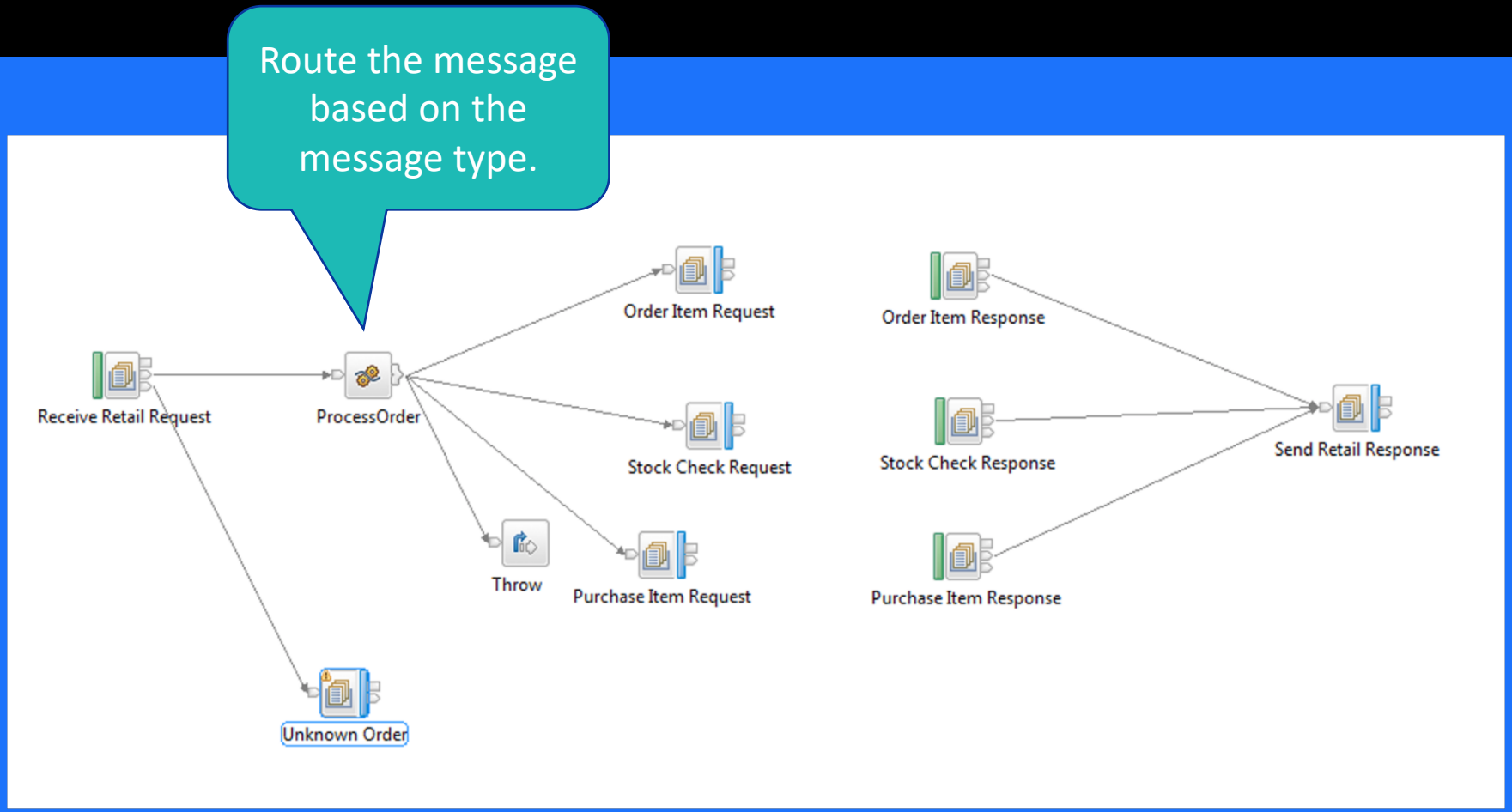
Replies are correlated with the original requests.

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

Use asynchronous flows to avoid bottlenecks.

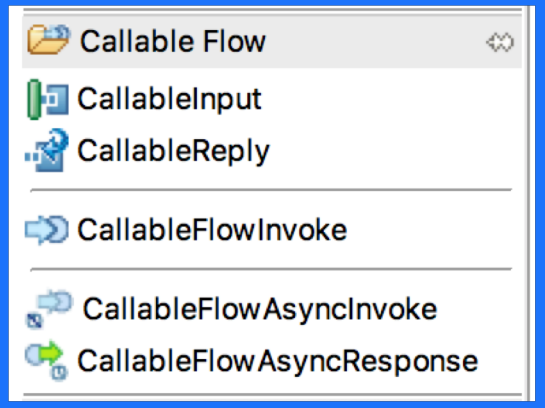
Message Flow Structures

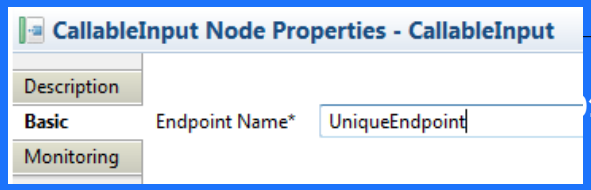
- Separation of processing
  - Split out incoming different requests received at single entry point
  - Find the message type upfront quickly and then route accordingly.



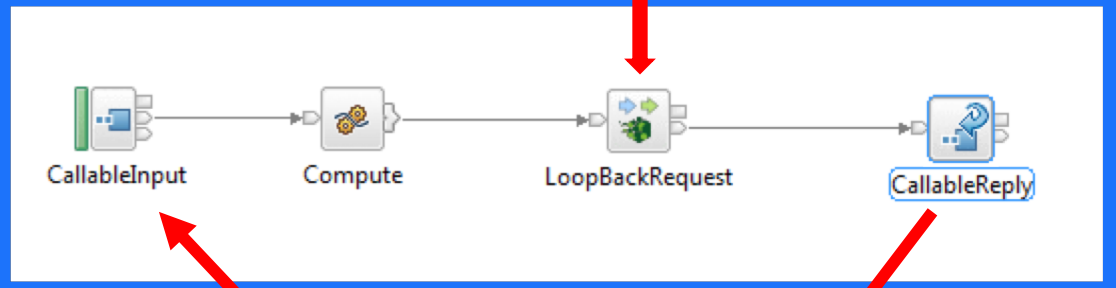
Message Flow Structures

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.

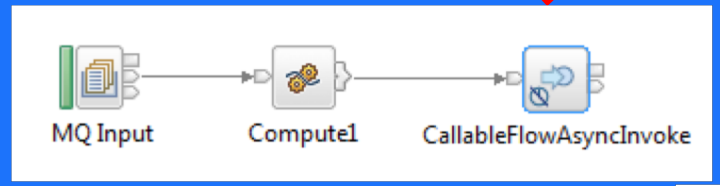




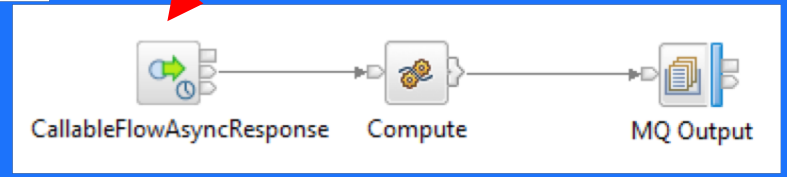
WINDOWS



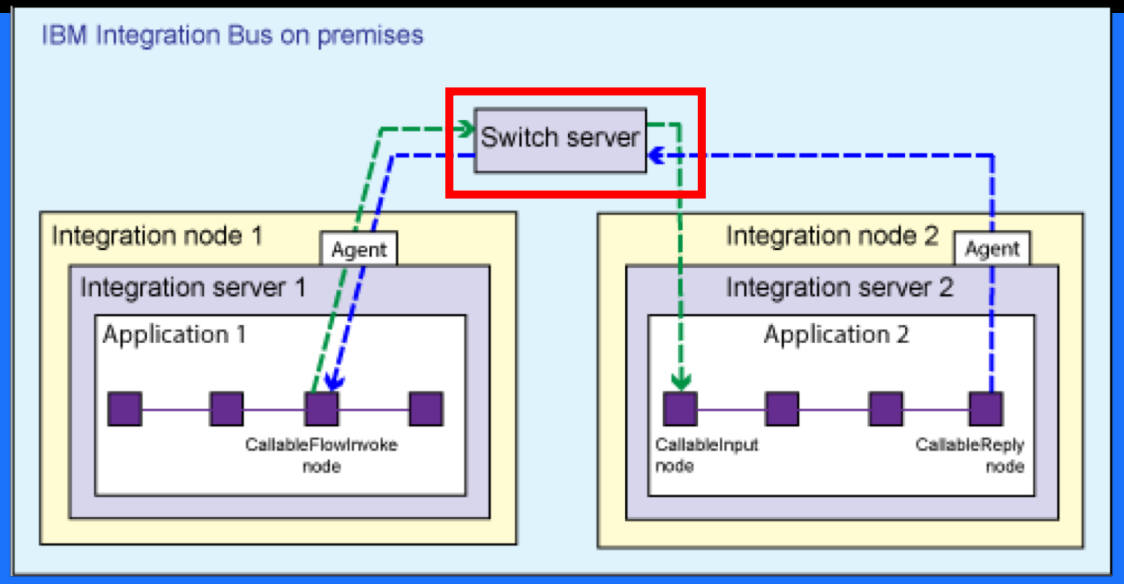
Use asynchronous flows to avoid bottlenecks.

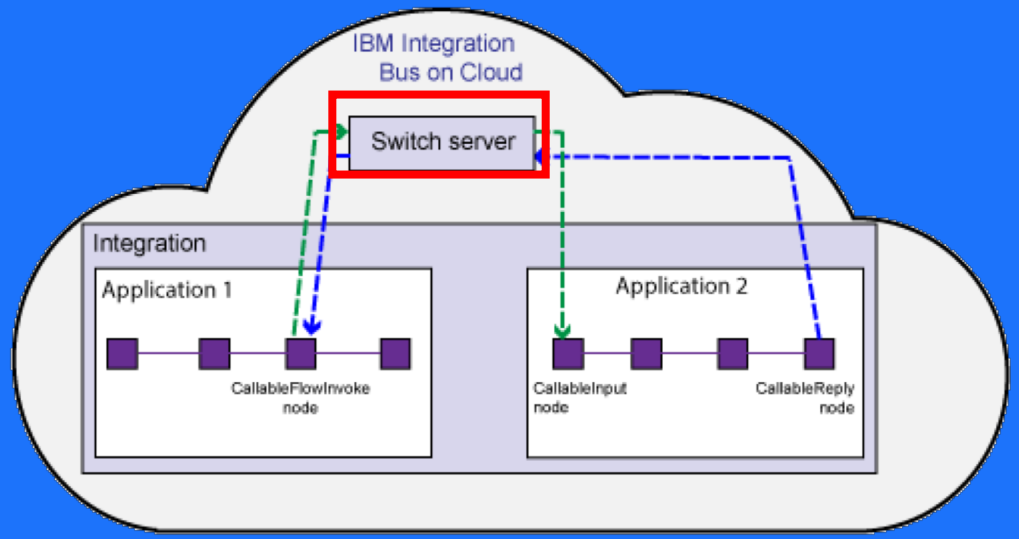


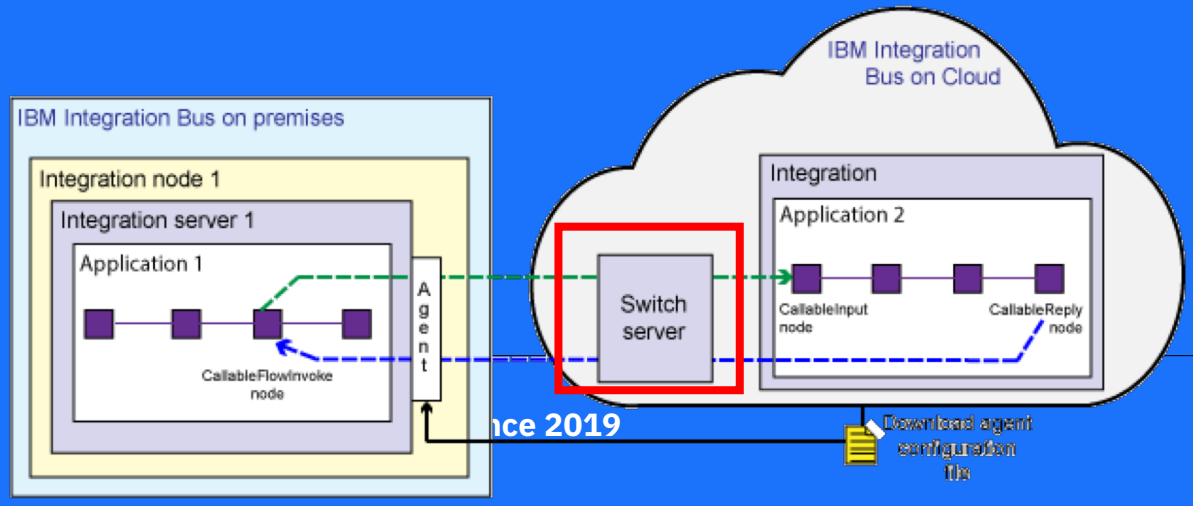
LINUX

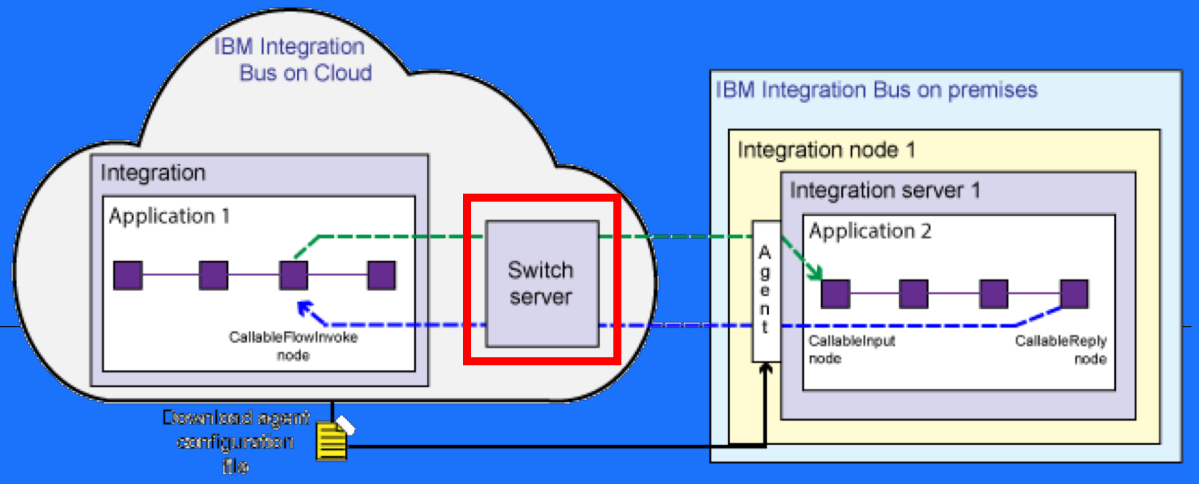


Message Flow Structures



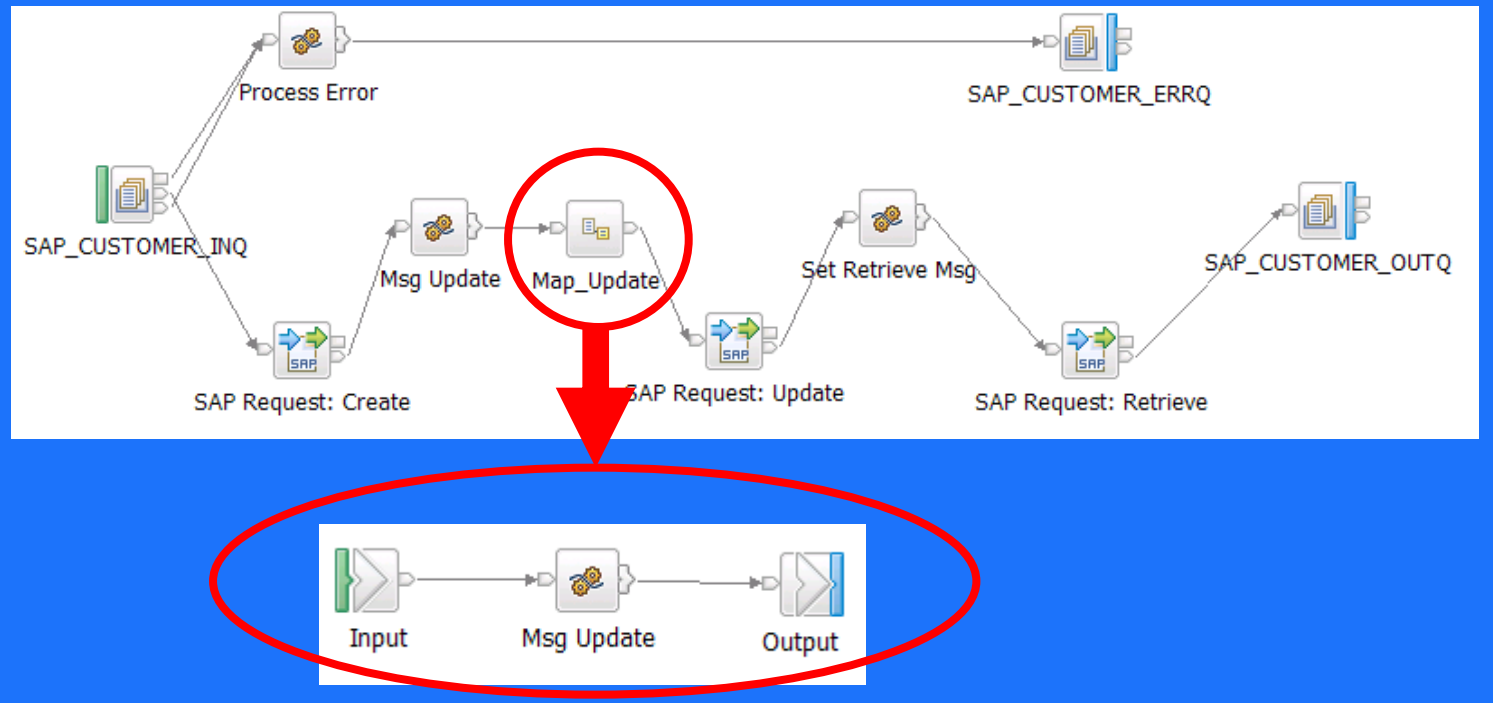






Subflows

- 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



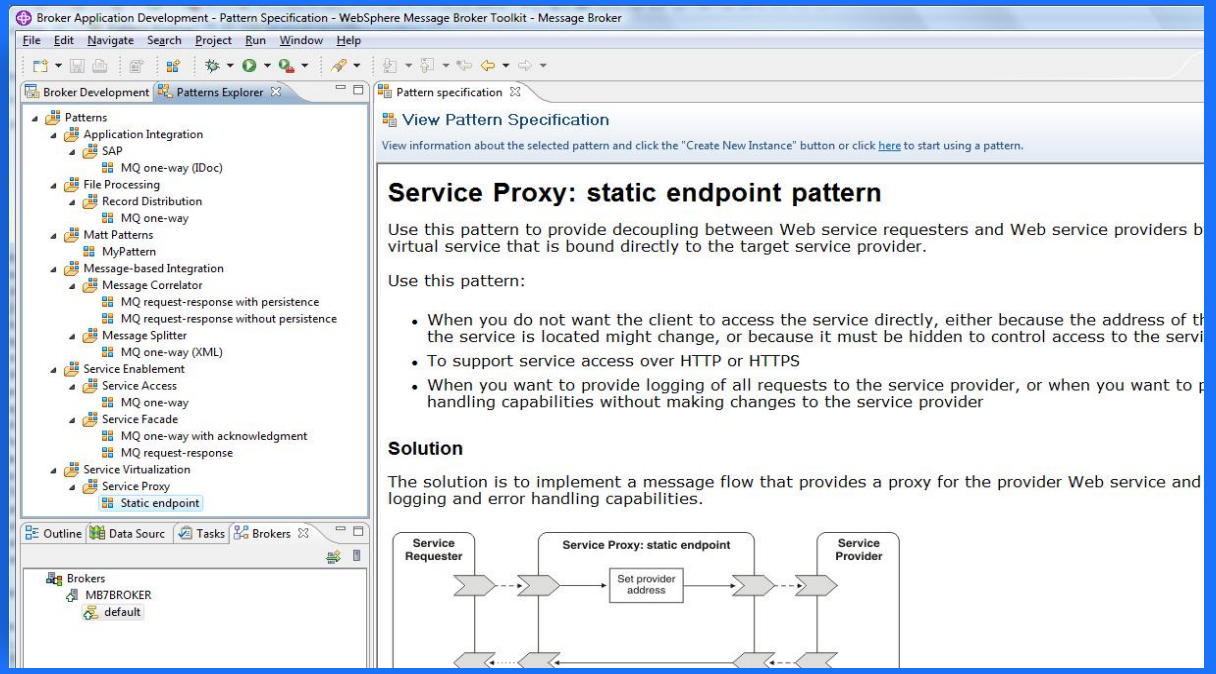
When it is Right Make a Pattern of It

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

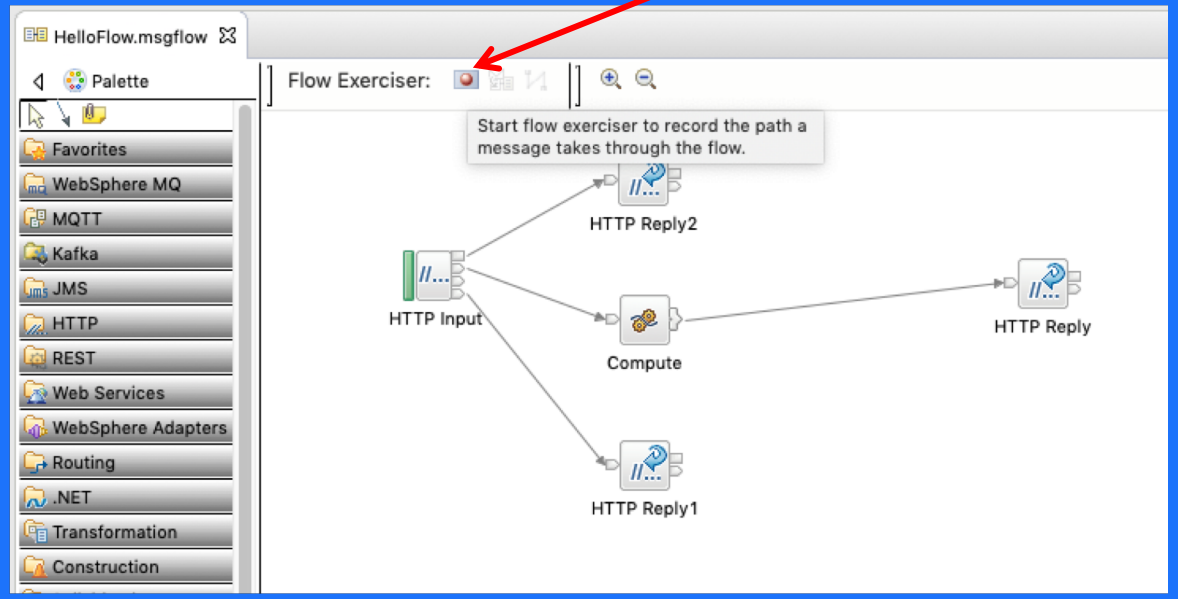
- Reduce common problems in flow development
- Establish best practice
- Reduces time to develop solutions
- Helps new message flow developers to come on board more quickly.



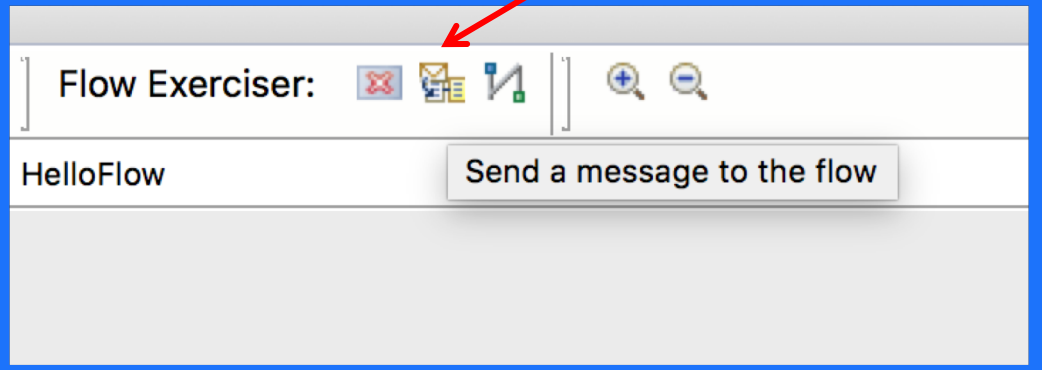
Test Considerations

Flow Exerciser – use to test your flows

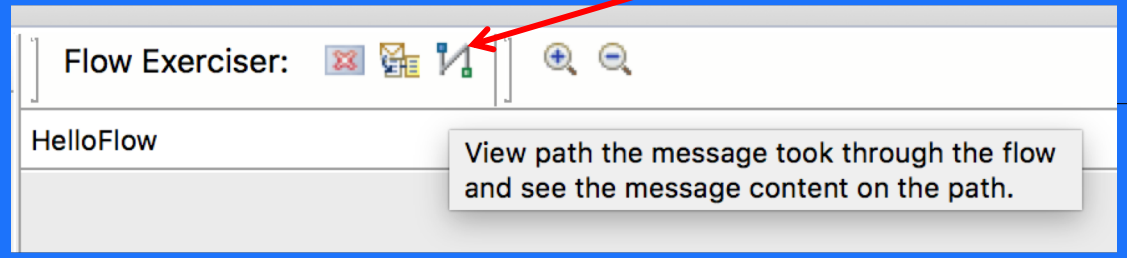
Recording: Start the flow exerciser

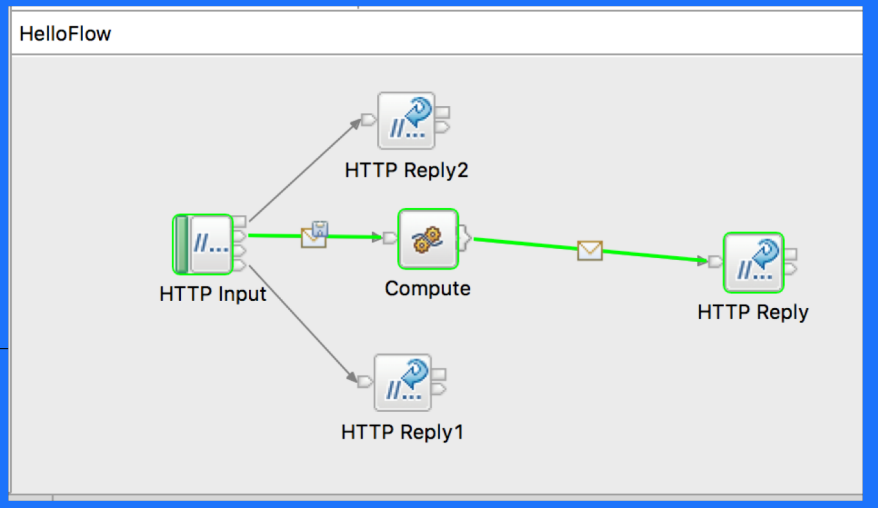


Injection: Send a message to the flow

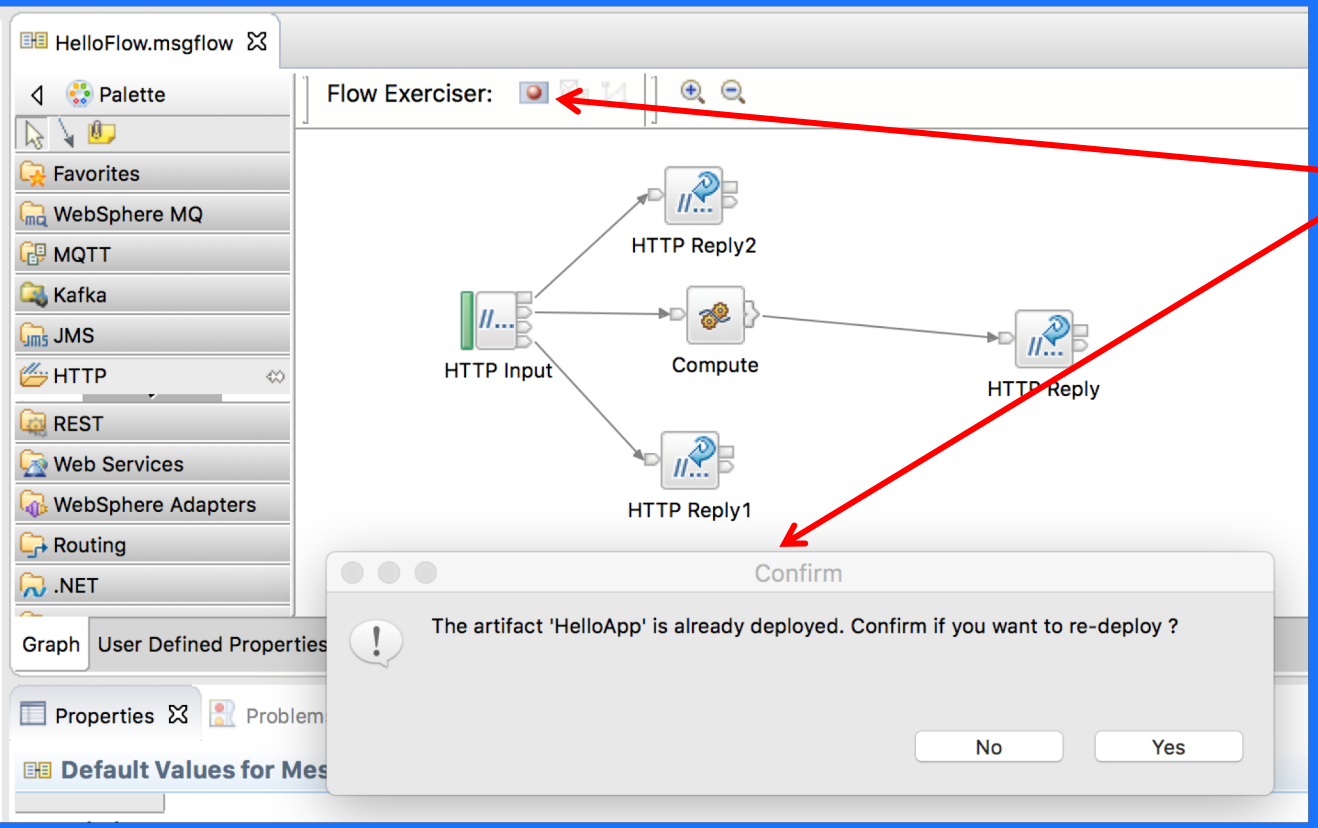


Viewing: Viewing the message processing path and message content



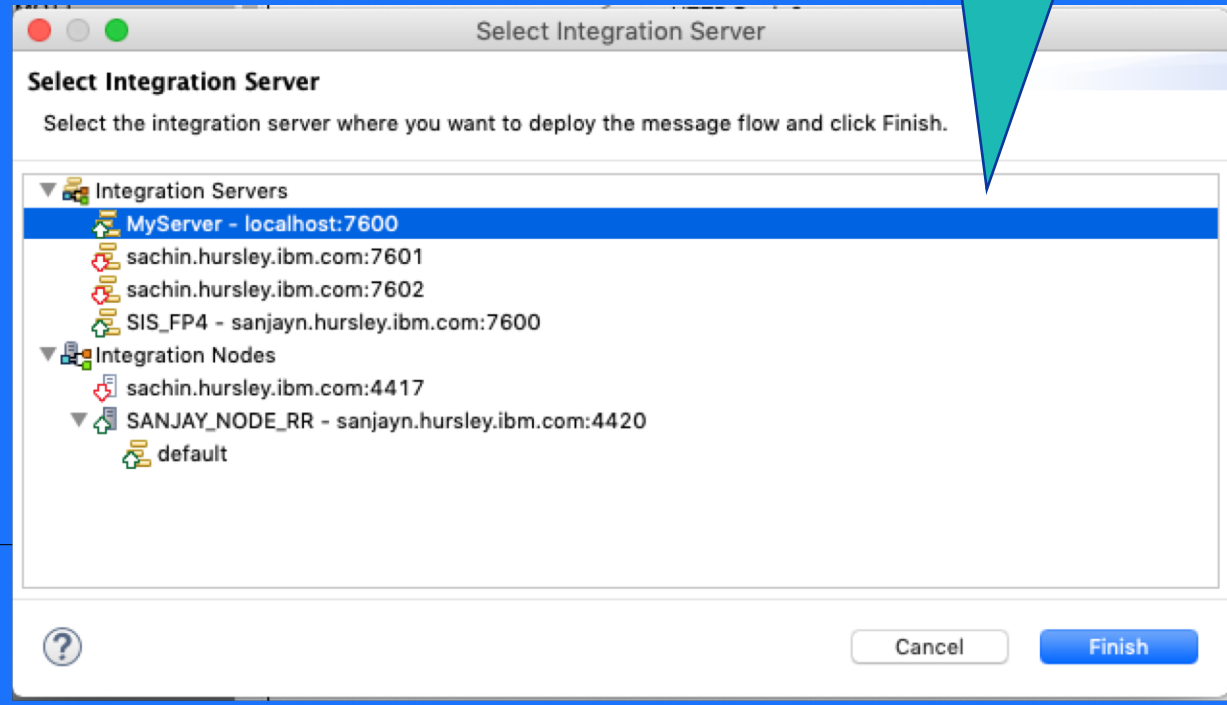


Flow Exerciser – recording a message

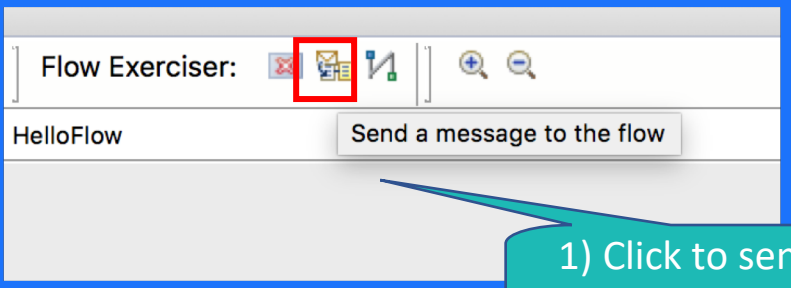


New in ACE V11, a confirmation dialog is shown to re-deploy if the Application is already deployed.

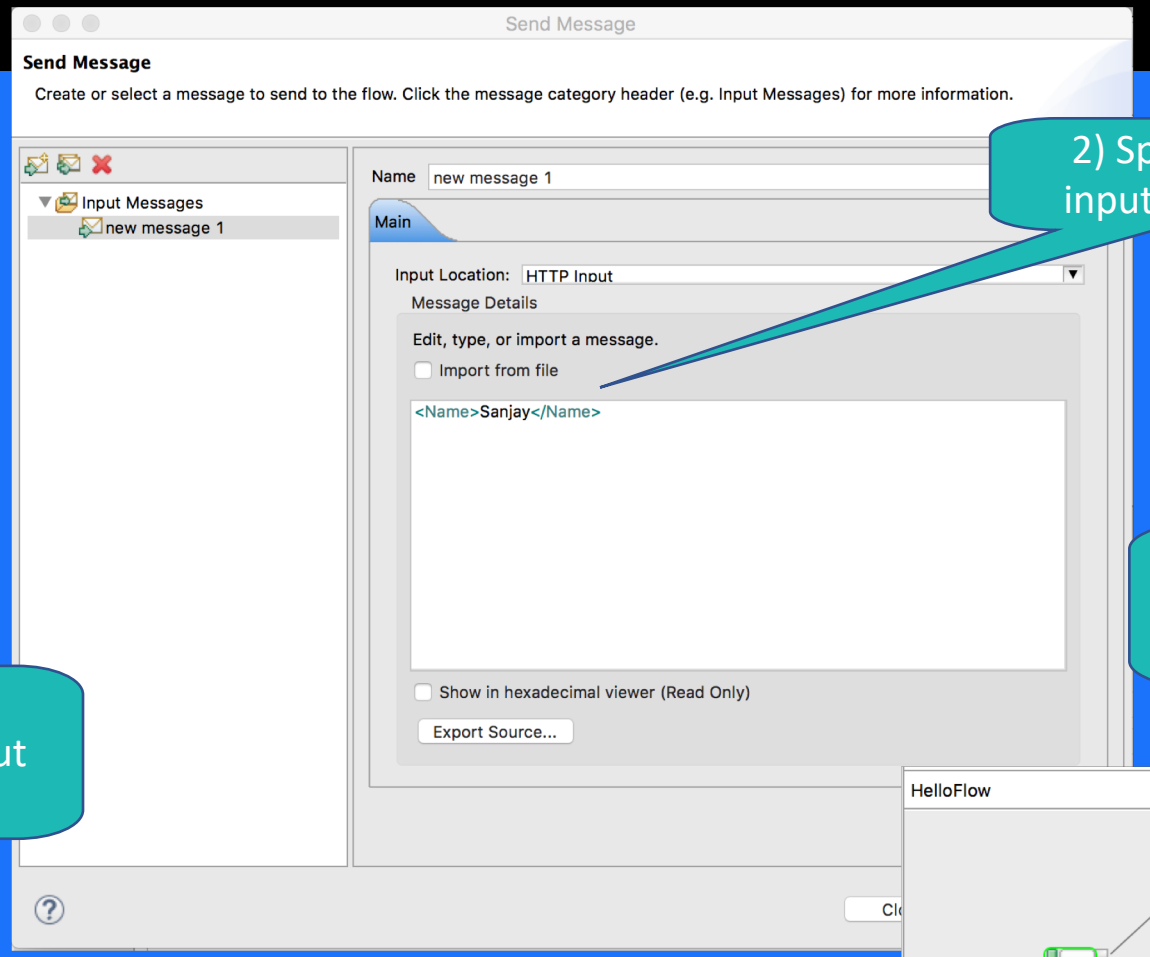
Choose a independent Integration Server or a node-owned Integration Server to deploy to.



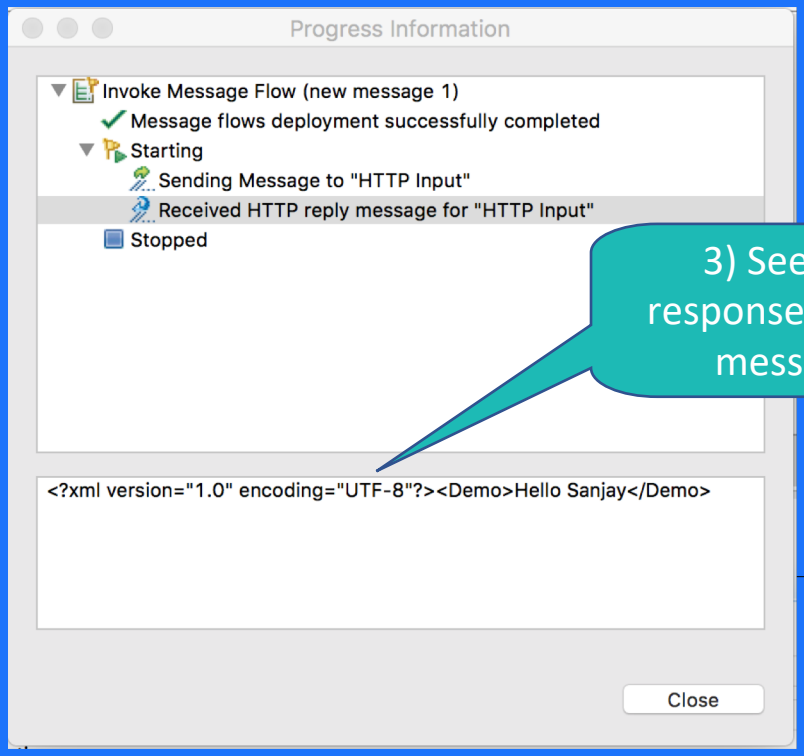
Flow Exerciser – sending a message



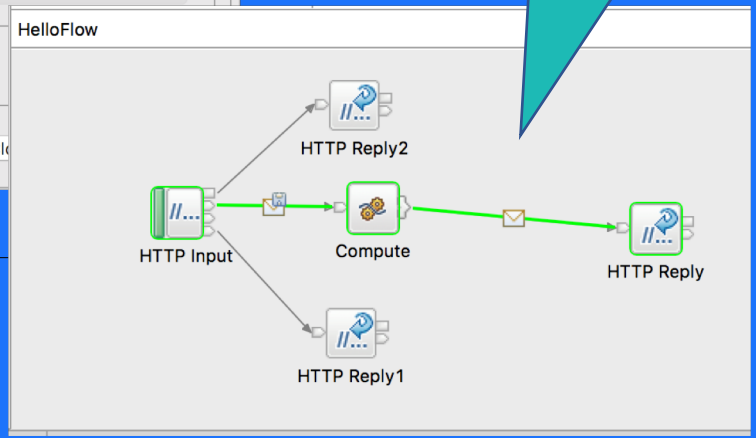
1) Click to send a message



2) Specify the input message

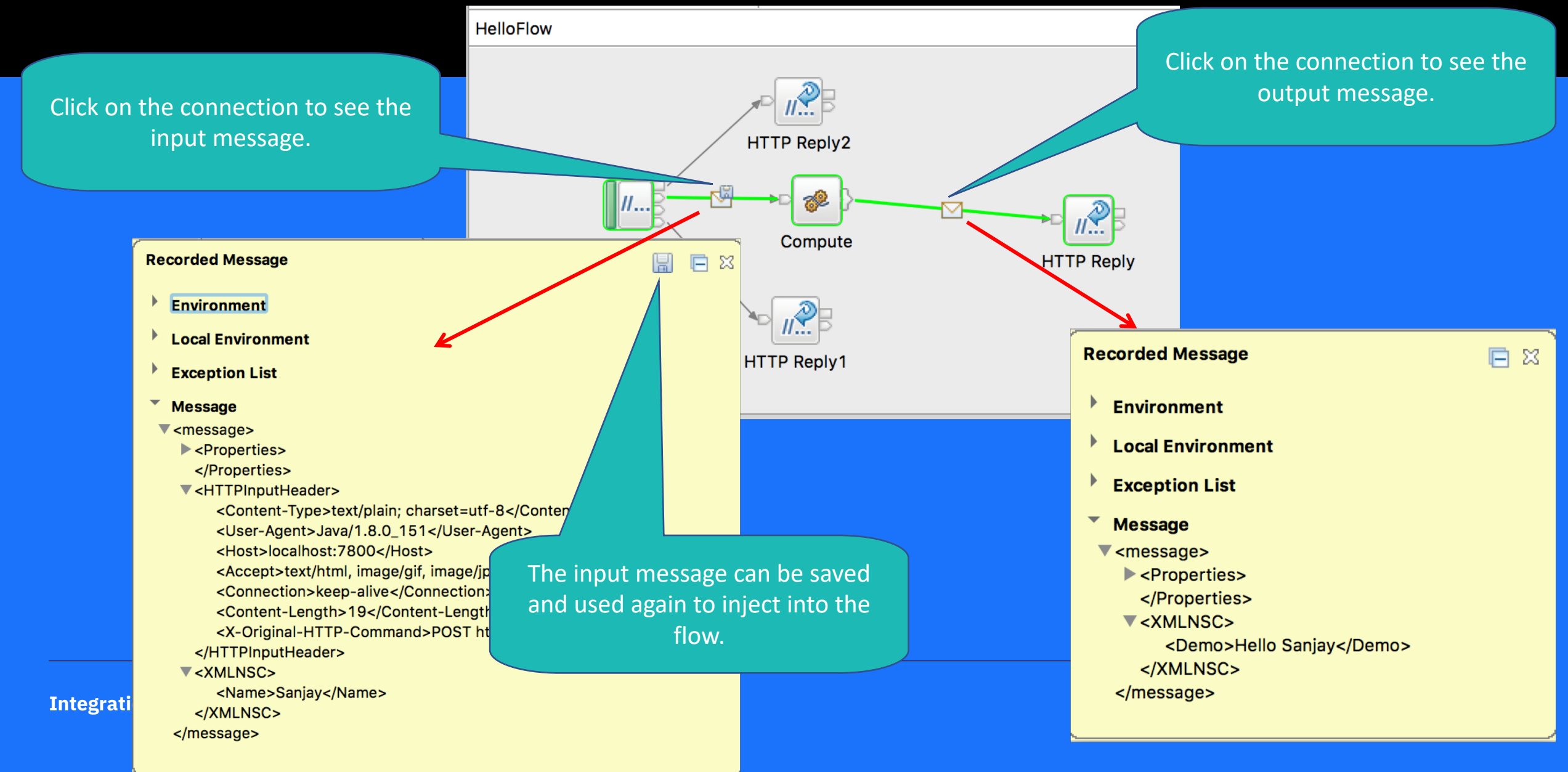


3) See the response output message

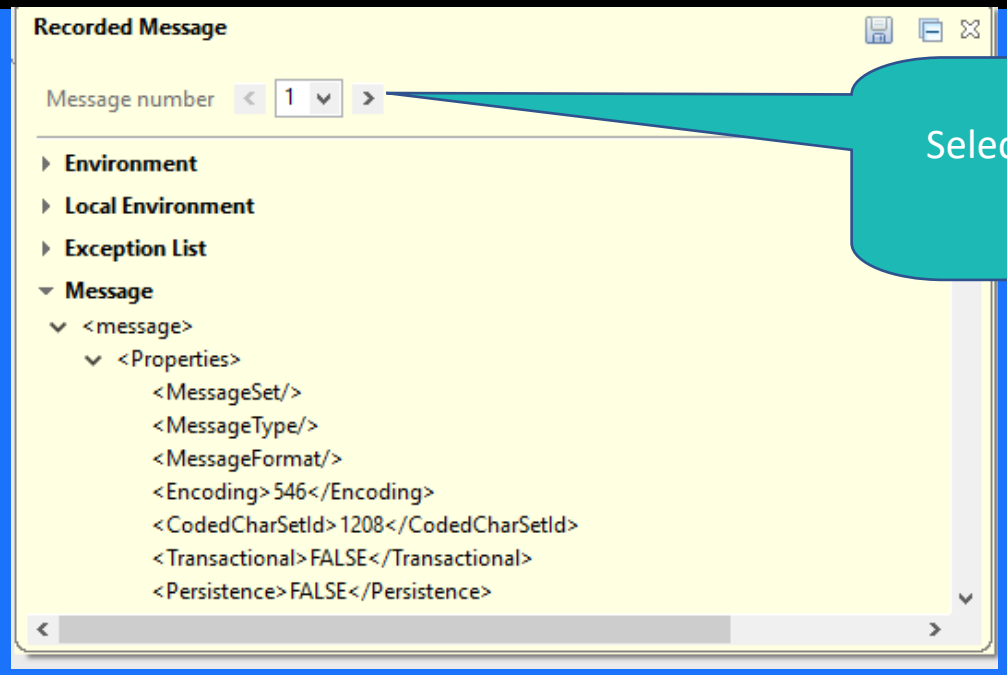


4) Path taken through the flow is highlighted.

Flow Exerciser – viewing a message and saving

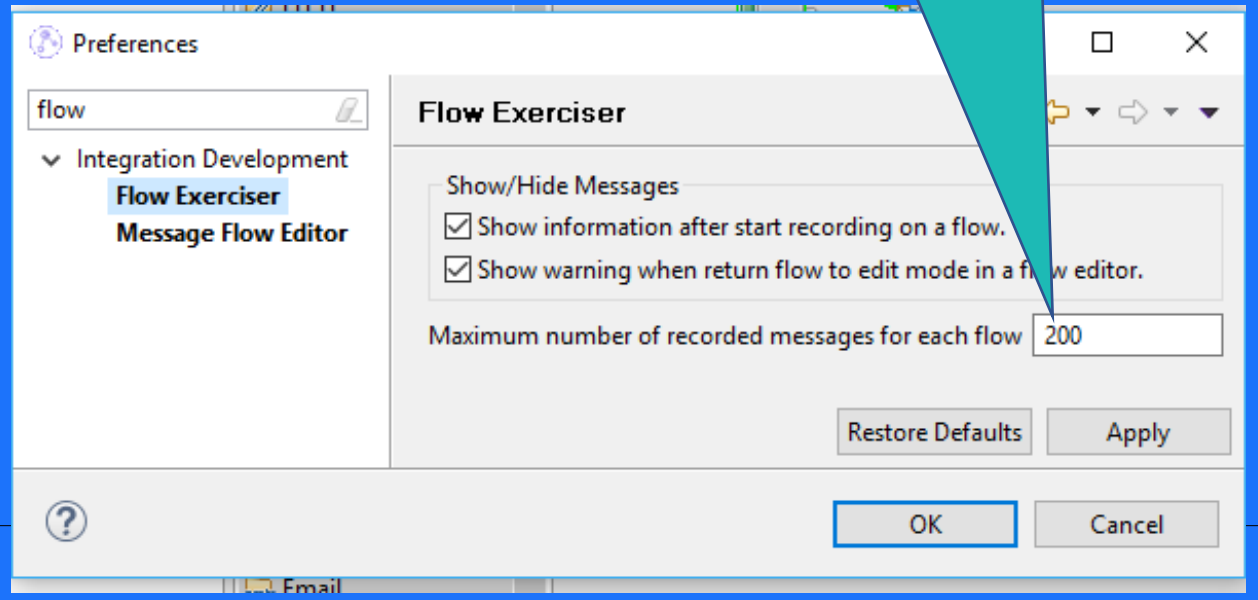


Flow Exerciser – viewing multiple messages



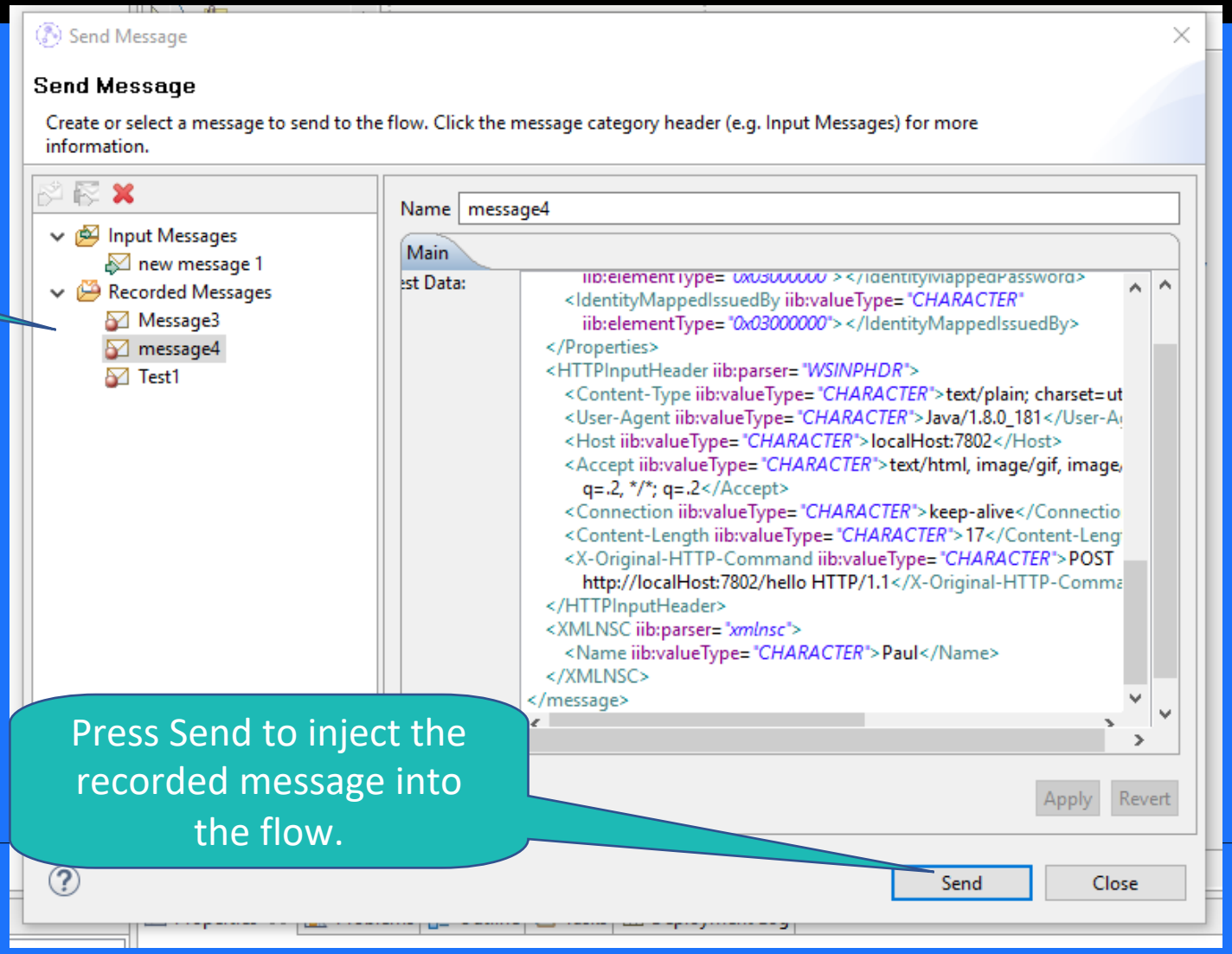
Select the message you want to see.

You can limit the amount of recorded messages that are returned.



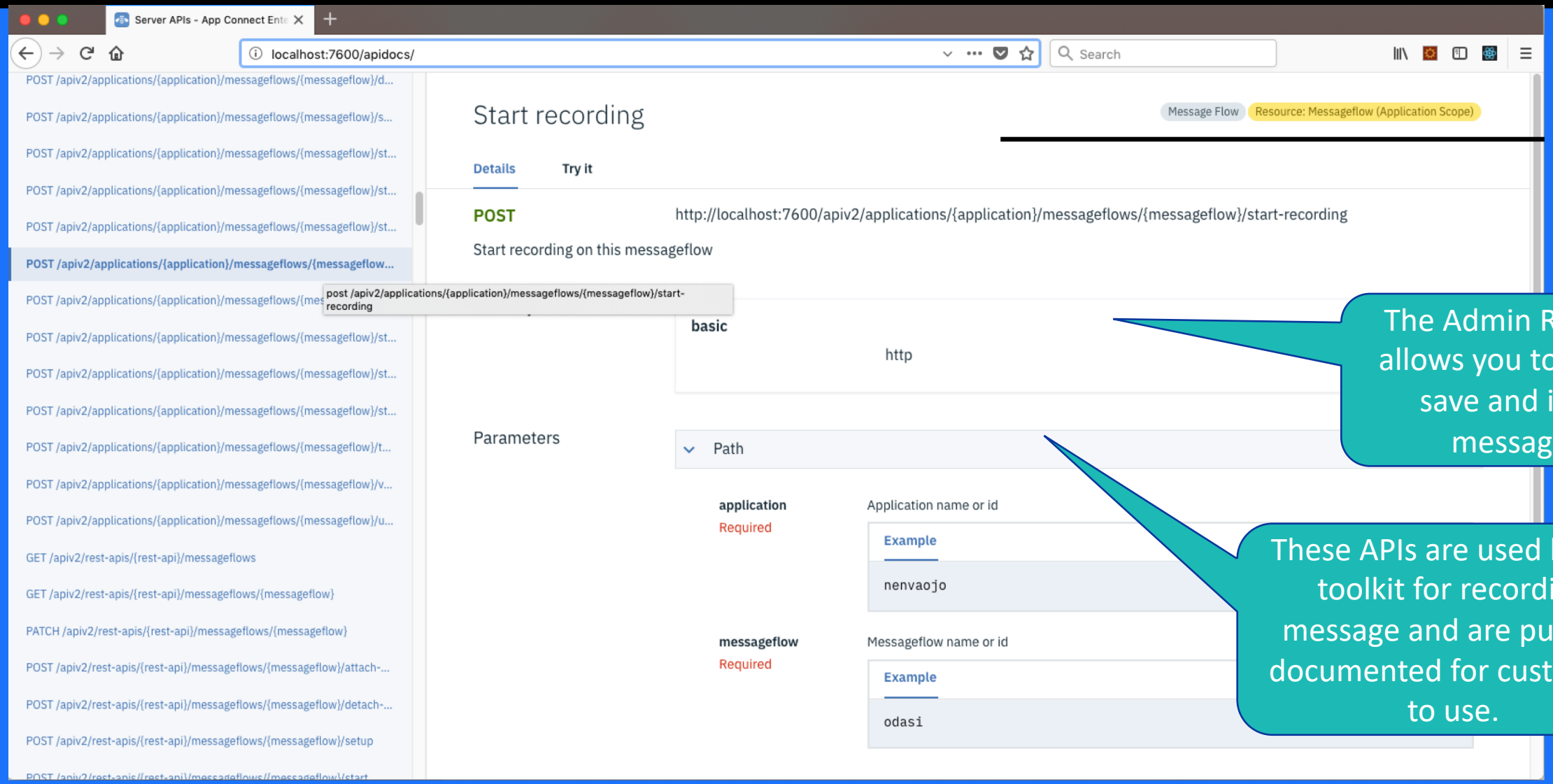
Flow Exerciser – injecting a recorded message

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



Press Send to inject the recorded message into the flow.

Flow Exerciser – using the Admin REST API



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

IBM

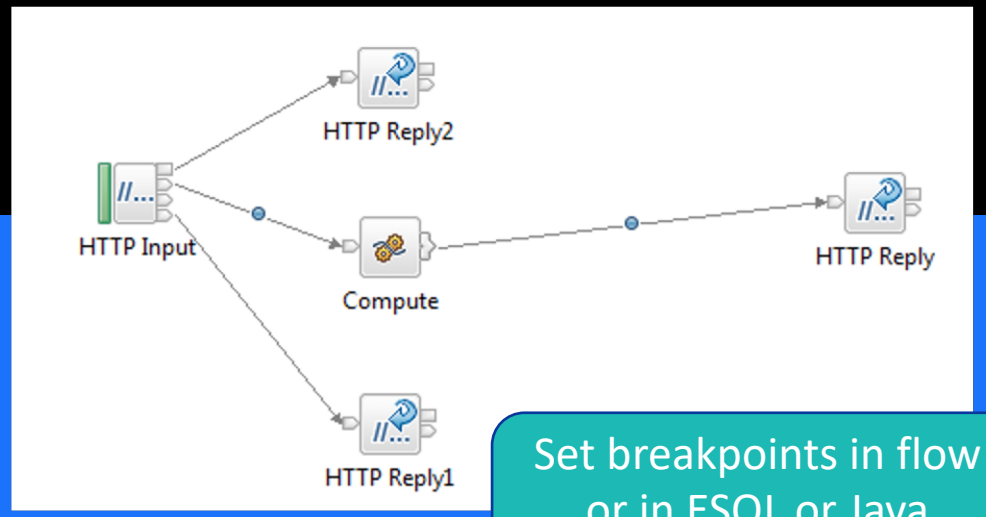

```
sanjays-mbp:~ sanjayn$ curl -X GET http://localhost:7600/apiv2/data/recorded-test-data?depth=6 | python3 -m json.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 22847  100 22847    0     0  3100k      0 --:--:-- --:--:-- --:--:-- 3187k
{
    "count": 6,
    "recordedTestData": [
        {
            "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": "ComIbmWSInputNode",
                            "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"
                }
            },
            "testData": {
                "message": "PG1lc3NhZ2UgeG1sbnM6aWliPSJodHRwOi8vY29tLmlibS5paWIvbHQvMS4wIiBpaWI6cGFyc2VyPSJXU1Jvb3QiIG
[illegible]
U0MgaWliOnBhcnNlcj0ieG1sbnNjIj48TmFtZSBpaWI6dmFsdWVUeXBlPSJDSEFSQUNURVIiPlNhbmpheTwvTmFtZT48L1hNTE5TQz48L21lc3NhZ2U+",
                "localEnvironment": "PGxvY2FsRW52aXJvbm1lbnQgeG1sbnM6aWliPSJodHRwOi8vY29tLmlibS5paWIvbHQvMS4wIiBpaWI6c
1QiIGlpYjppbmplY3RhYmxlPSJ0cnVlIj48RGVzdGluYXRpb24+PEhUVFA+PFJlcXVlc3RJZGVudGlmaWVyIGlpYjp2YWx1ZVR5cGU9IkJMT0IiPjQ1NTY
wMDAwMDAwMGVkZWQzMDRmMjczYTIxMDAwMDAwMDAwMDwvUmVxdWVzdElkZW50aWZpZXI+PC9IVFRQPjwvRGVzdGluYXRpb24+PC9sb2NhbEVudmlyb25tZW
                "environment": "",
                "exceptionList": ""
            }
        },
        {
            "checkpoint": {
                "messageFlowData": {
```


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/recorded-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

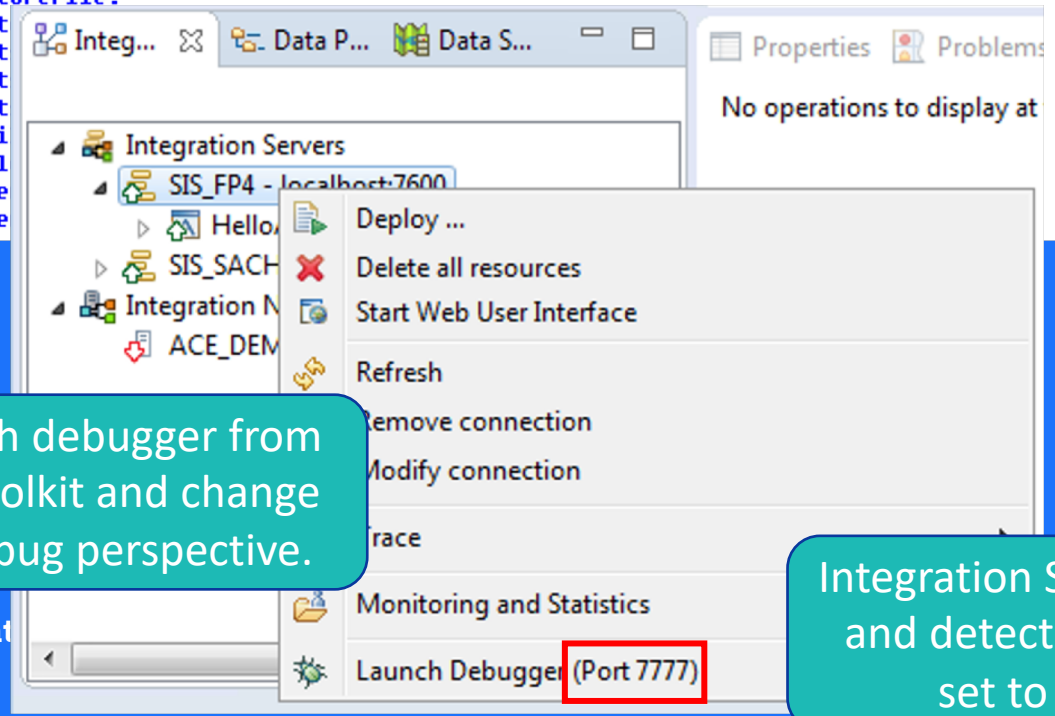
Flow Debugger

```
ResourceManagers:
  JVM:
    #jvmVerboseOption: 'none'
    #jvmDisableClassGC: ''
    #jvmEnableIncGC: ''
    #jvmShareClasses: ''
    #jvmNativeStackSize: -1
    #jvmJavaOSStackSize: -1
    #jvmMinHeapSize: 33554432          # minimum JVM  heap size in bytes (32MB)
    #jvmMaxHeapSize: 268435456         # 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



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



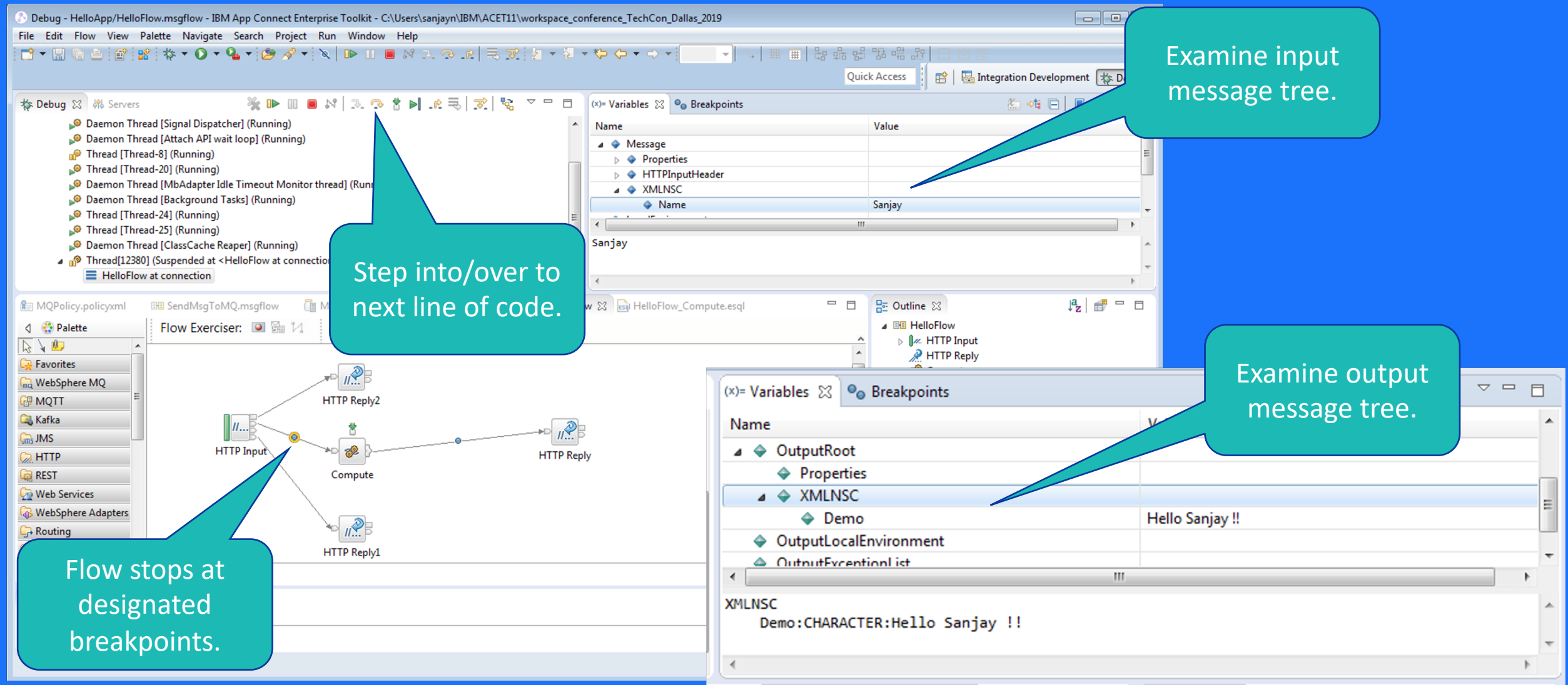
Launch debugger from the toolkit and change to debug perspective.

```
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 adm
ecurity is inactive.
:14:09.343780: The HTTP Listener has started listening on port '760
dmin http' connections.

:14:09.385678: Integration server has finished initialization.
```

Integration Server starts and detects JVM port set to 7777.

Flow Debugger



Testing a DFDL schema file

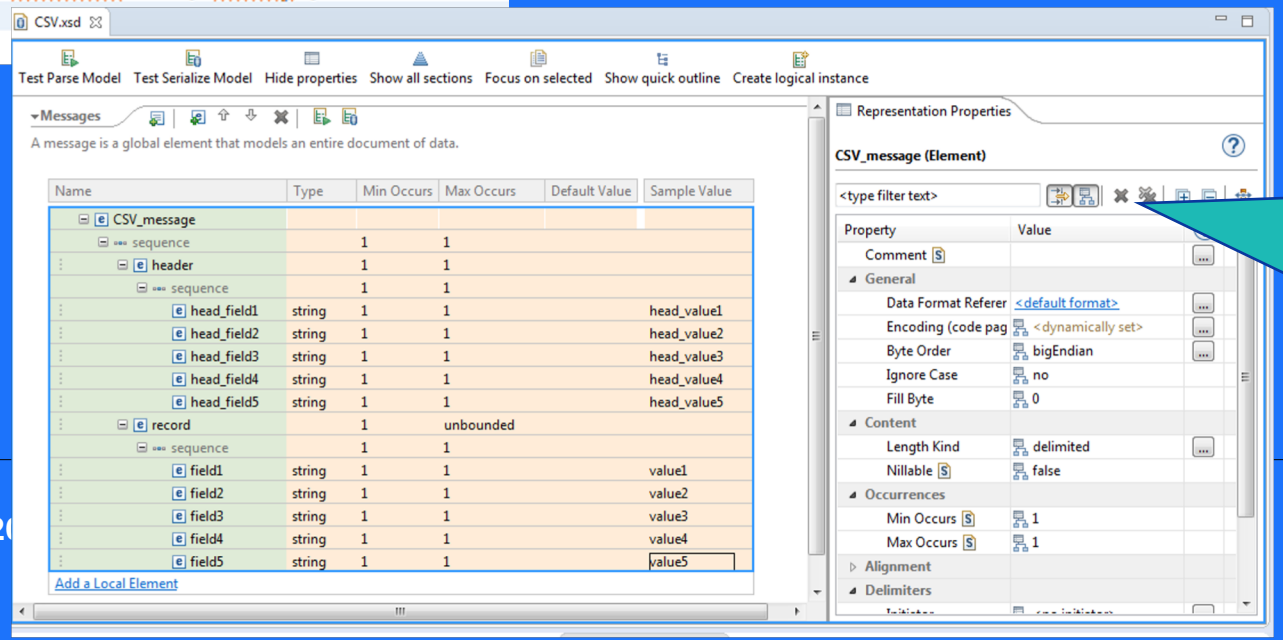
Non XML data (both text and binary) is defined using DFDL schemas

- 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

simpleCSV.txt

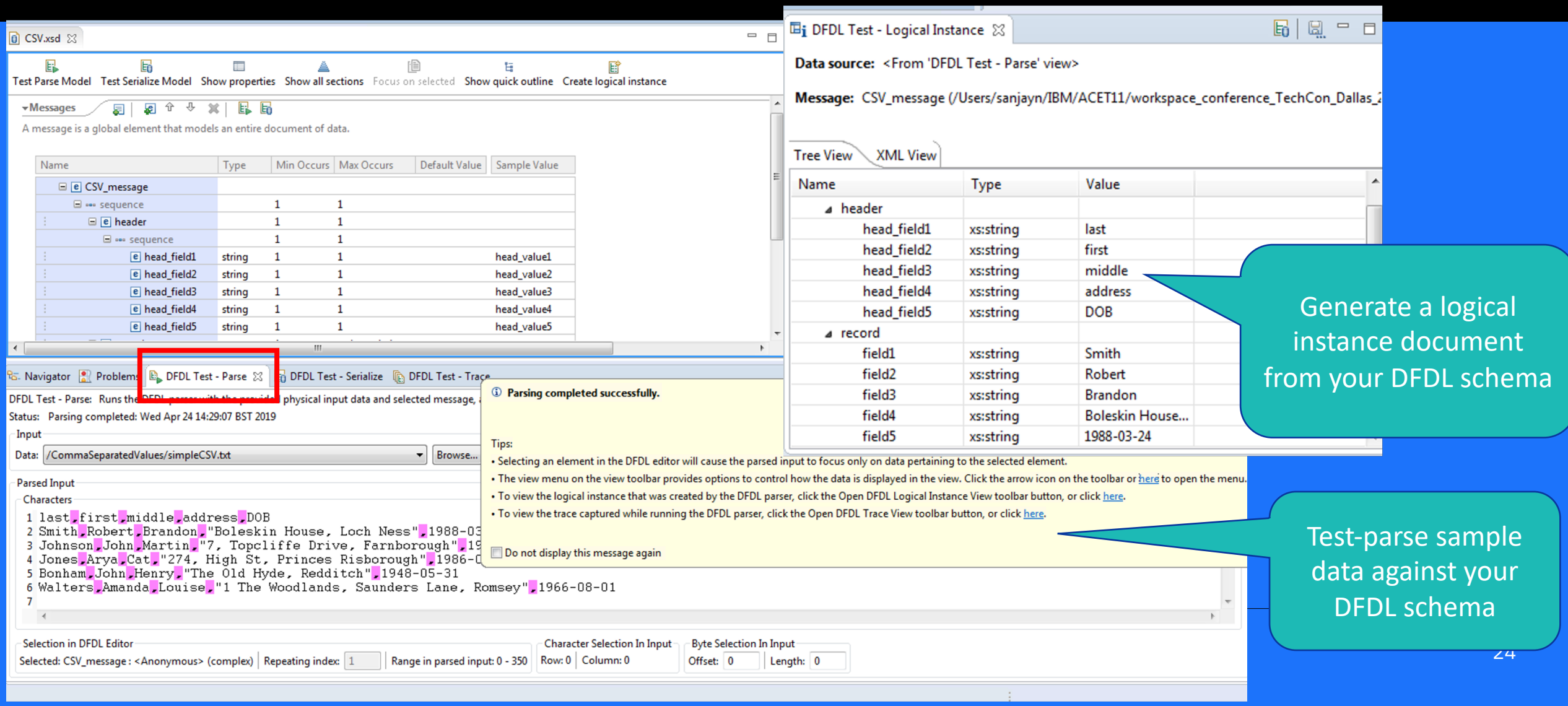
```
last,first,middle,address,DOB
Smith,Robert,Brandon,"Boleskin House, Loch Ness",1988-03-24
Johnson,John,Martin,"7, Topcliffe Drive, Farnborough",1986-01-23
Jones,Arya,Cat,"274, High St, Princes Risborough",1986-02-19
Bonham,John,Henry,"The Old Hyde, Redditch",1948-05-31
Walters,Amanda,Louise,"1 The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

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.

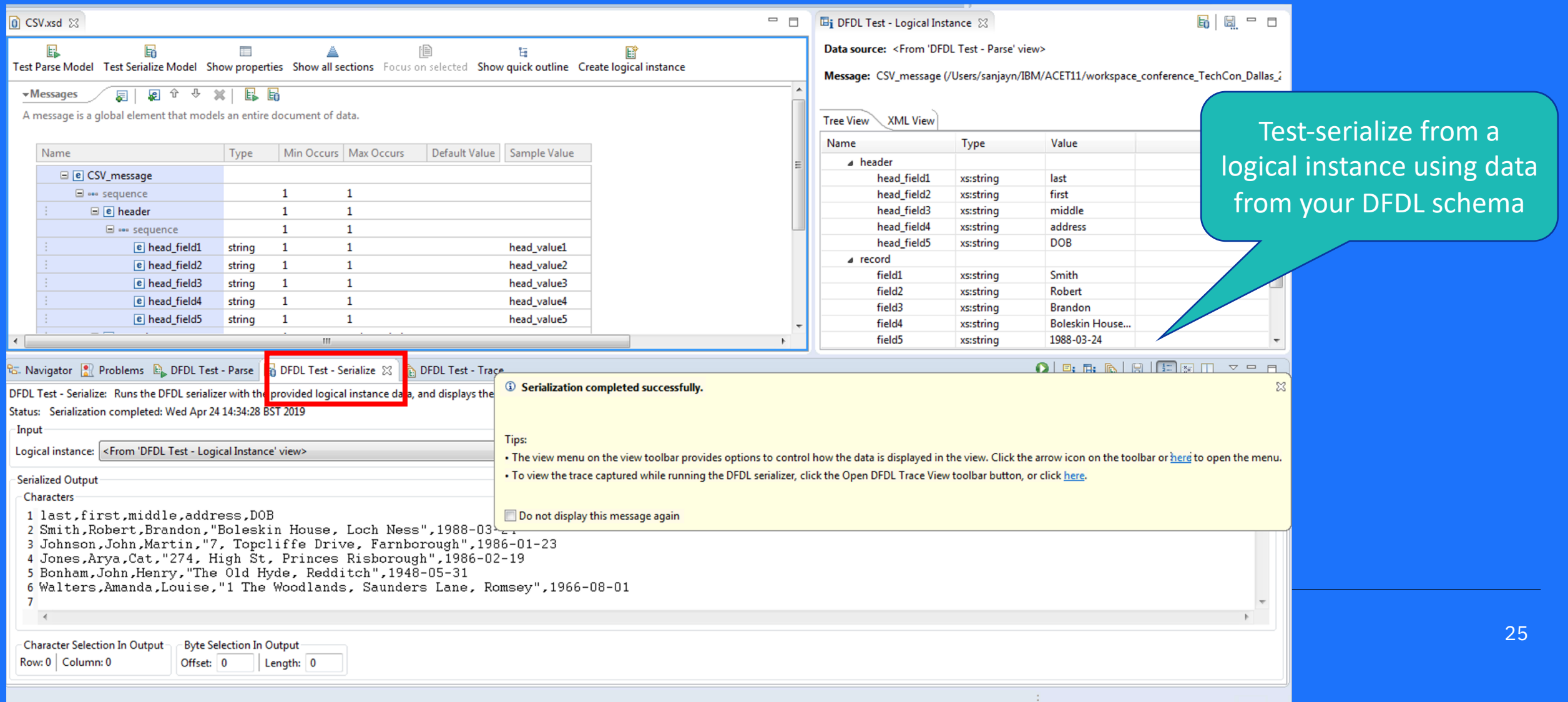
Testing a DFDL schema file



Generate a logical instance document from your DFDL schema

Test-parse sample data against your DFDL schema

Testing a DFDL schema file



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!)
- Performance evaluation and improvement is an iterative process:
  - Design
  - Code
  - Run
  - Measure
  - Identify hotspots
- Important to have the philosophy, process, infrastructure and tools to enable this

Start with a single instance if the flow on the server

Test flows in isolation

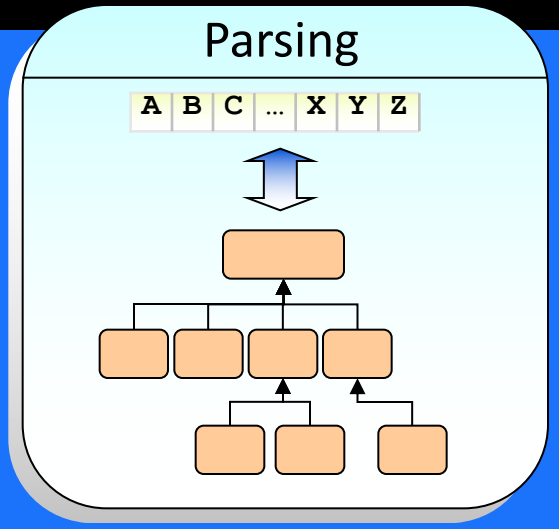
Take a baseline before changes are made

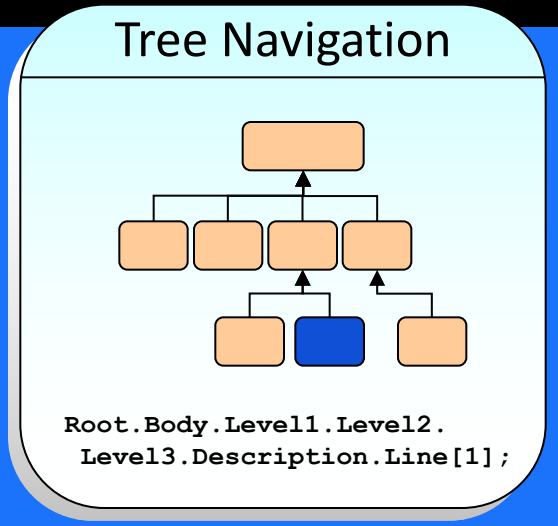
Make ONE change at a time and re-test to measure impact

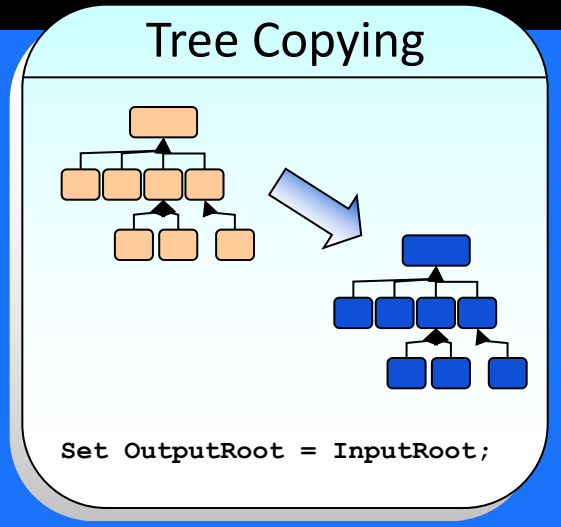
Run a driver application separate to your ACE machine

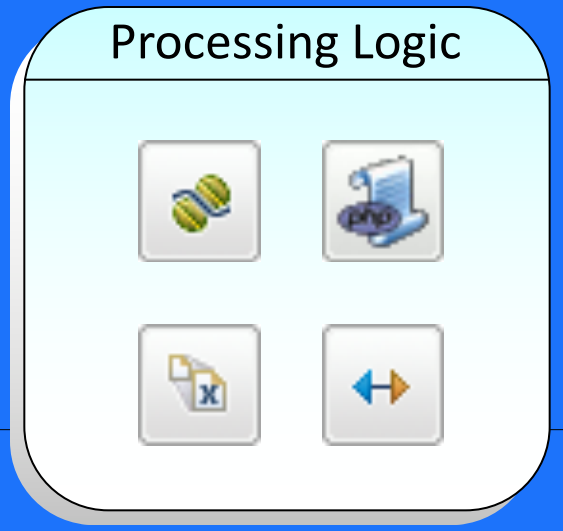
Use real data and avoid pre-loading queues.

What are the main performance costs in message flows?

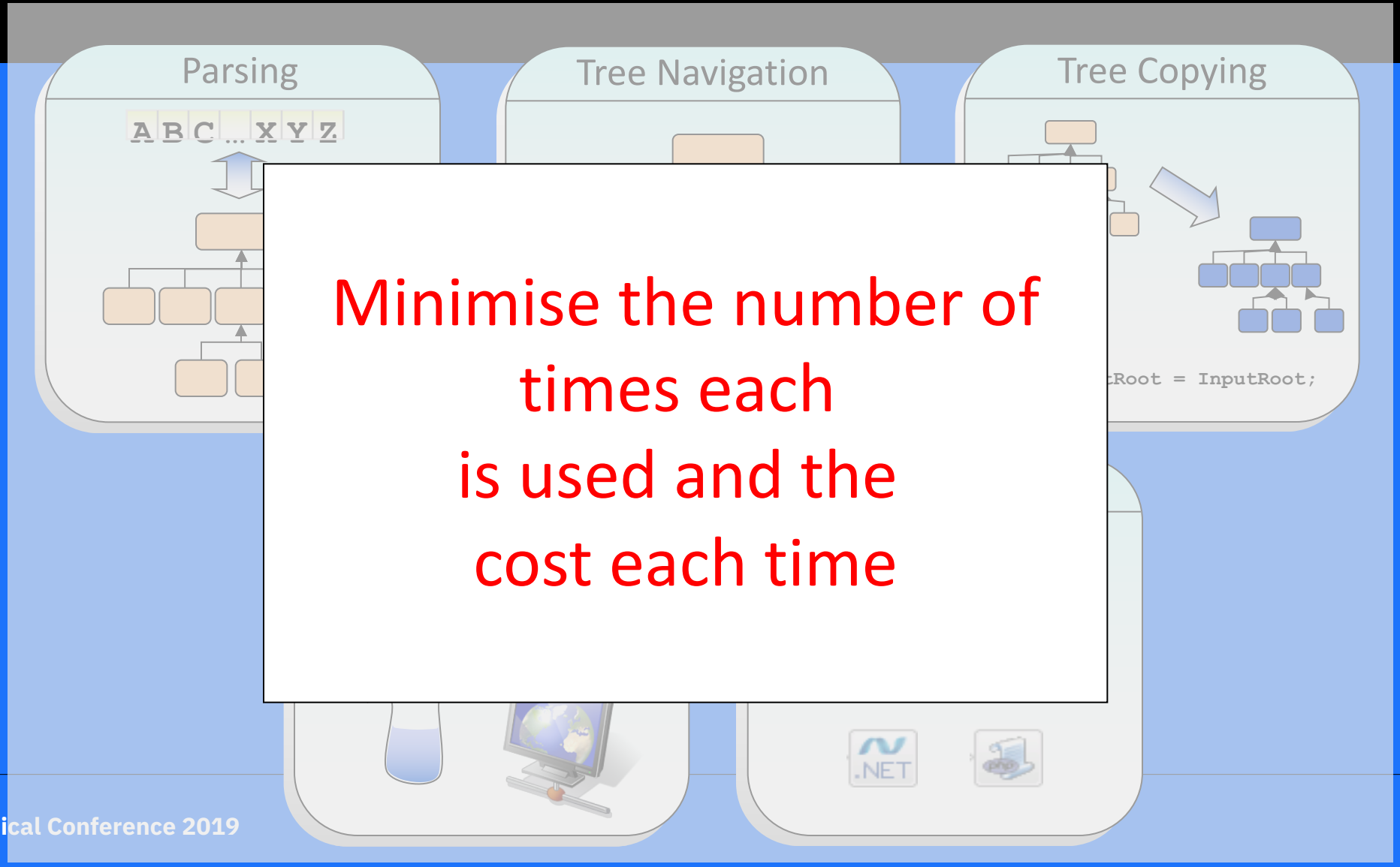






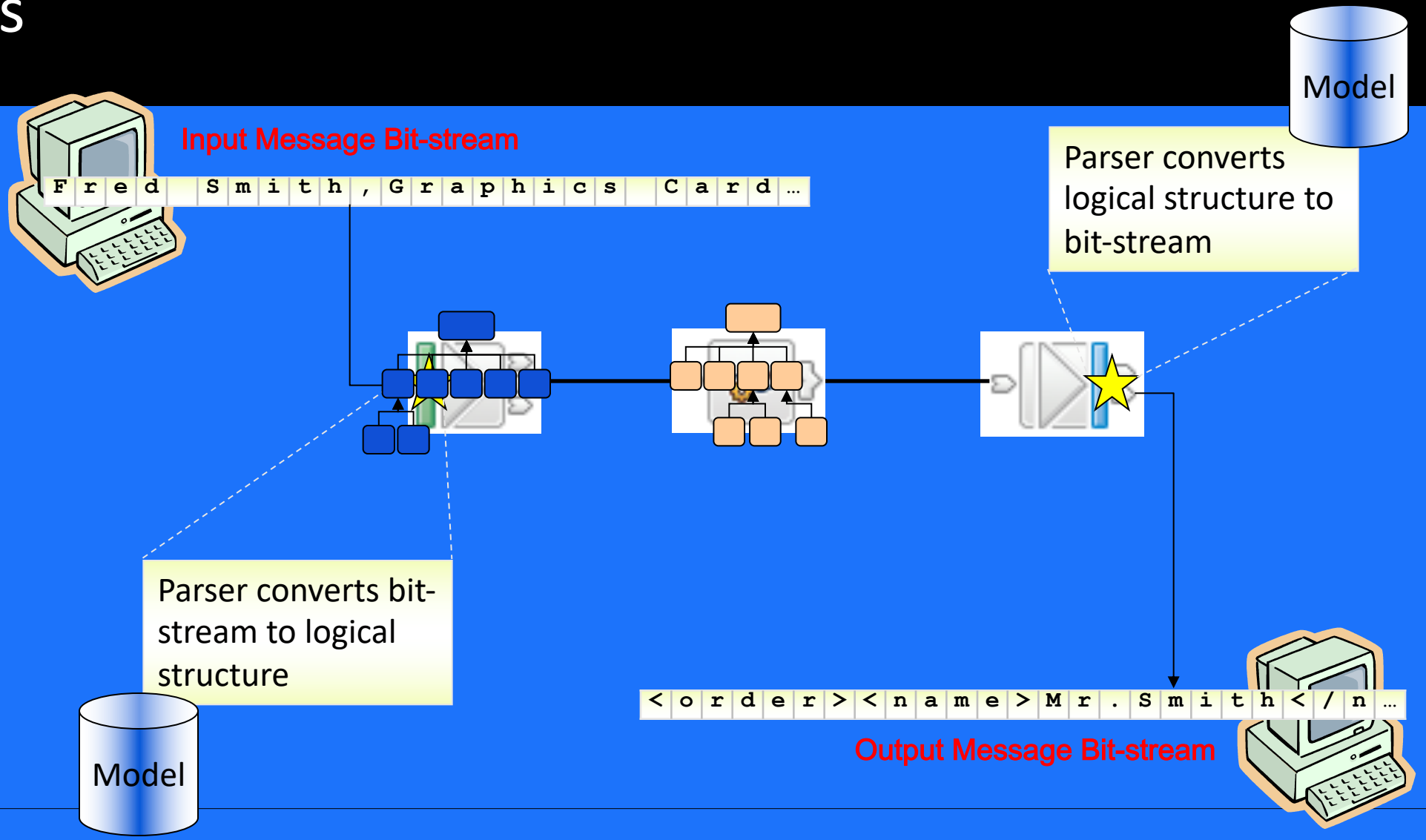


Techniques to Help Optimise Performance



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

Parsers


Model
Input Message Bit-stream
F r e d S m i t h , G r a p h i c s C a r d …
Parser converts logical structure to bit-stream
Parser converts bit-stream to logical structure
Model
< o r d e r > < n a m e > M r . S m i t h < / n …
Output Message Bit-stream

Parsing

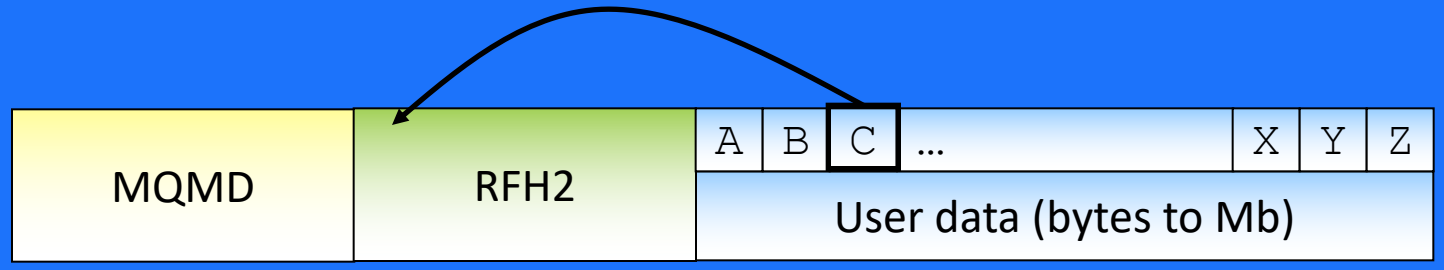
- The means of populating and serializing the tree
  - Multiple Parsers available: XMLNSC, JSON, DFDL, MIME, JMSMap, JMSStream, BLOB, SOAP
  - A Parser may use a model:
    - Parse a message that is not self-defining (CSV, COBOL, C messages)
    - Validate a message
    - Used for developing transformation 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

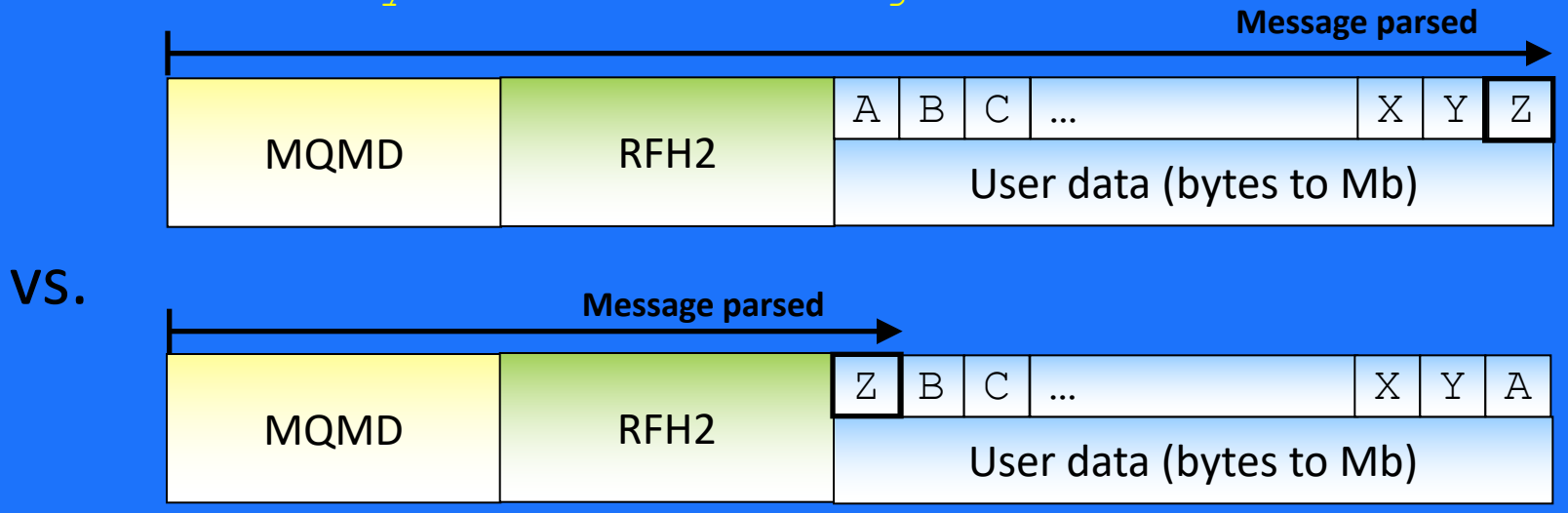
- If possible, avoid the need to parse at all!
  - Consider only sending changed data
  - Promote/copy key data structures to MQMD, MQRFH2 or JMS Properties
    - May save having to parse the user data
    - Particularly useful for message routing



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

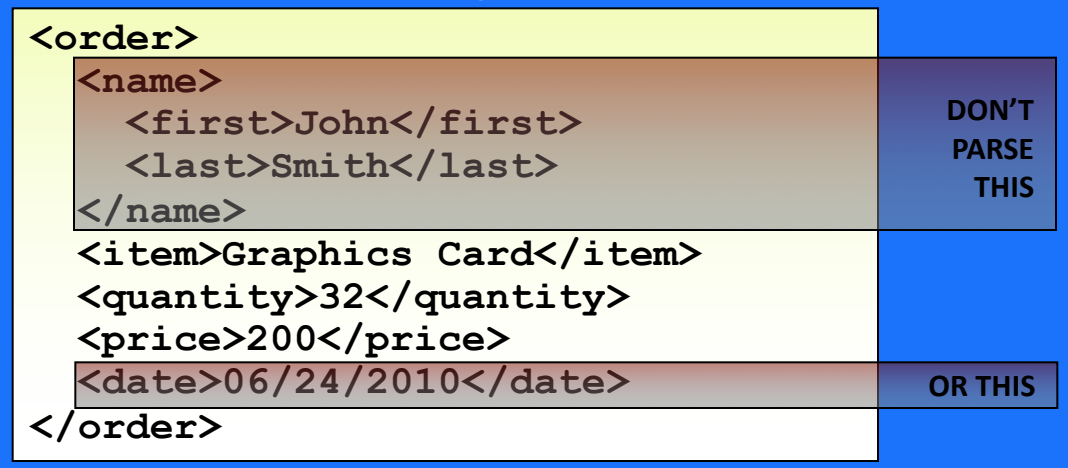
```
Set MyVariable = <Message Element Z>
```



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

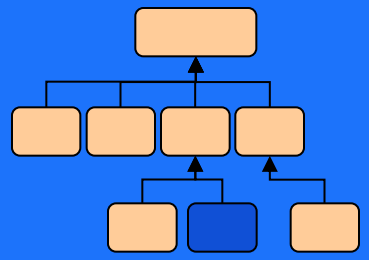
Opaque parsing

- 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!

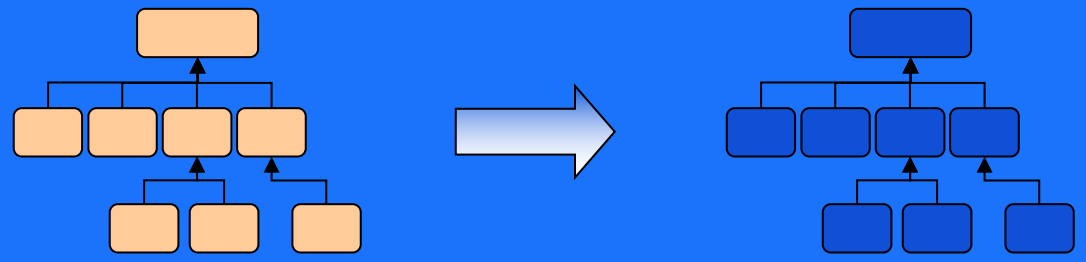


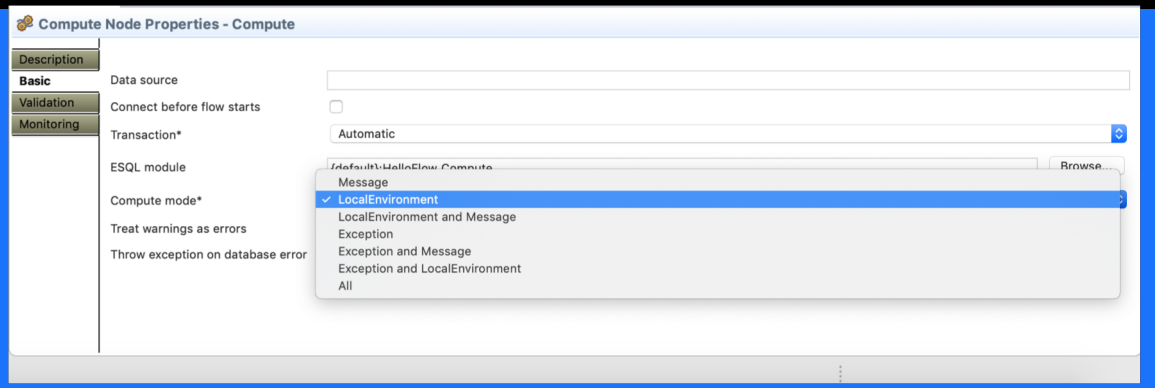
```
SET Description =
        Root.Body.Level1.Level2.Level3.Description.Line[1];
```

- 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)

Resource Access

MQ

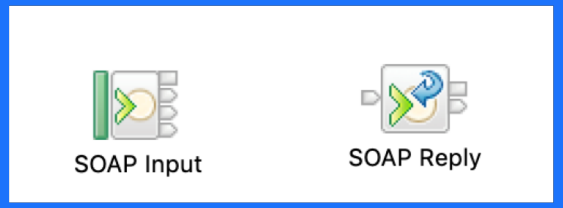
- Tune the QM and associated QM's, logs, buffer sizes
- Avoid unnecessary use of persistent messages,
- But use persistent messages as part of a co-ordinated transaction
- Use fast storage if using persistent messages

JMS

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

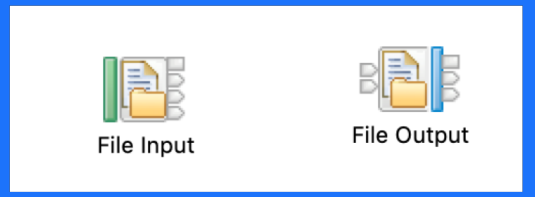
SOAP/HTTP

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



File

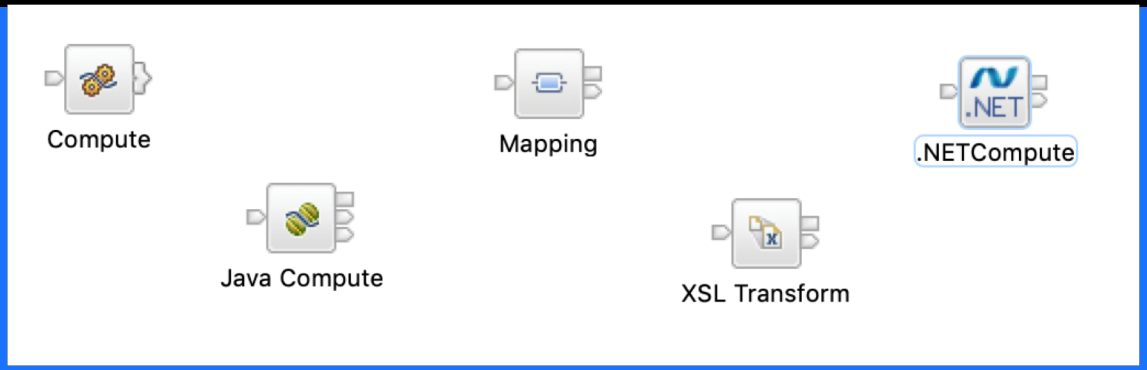
- Use cheapest delimiting option (see reports)
- Use fast storage



Transformation Options

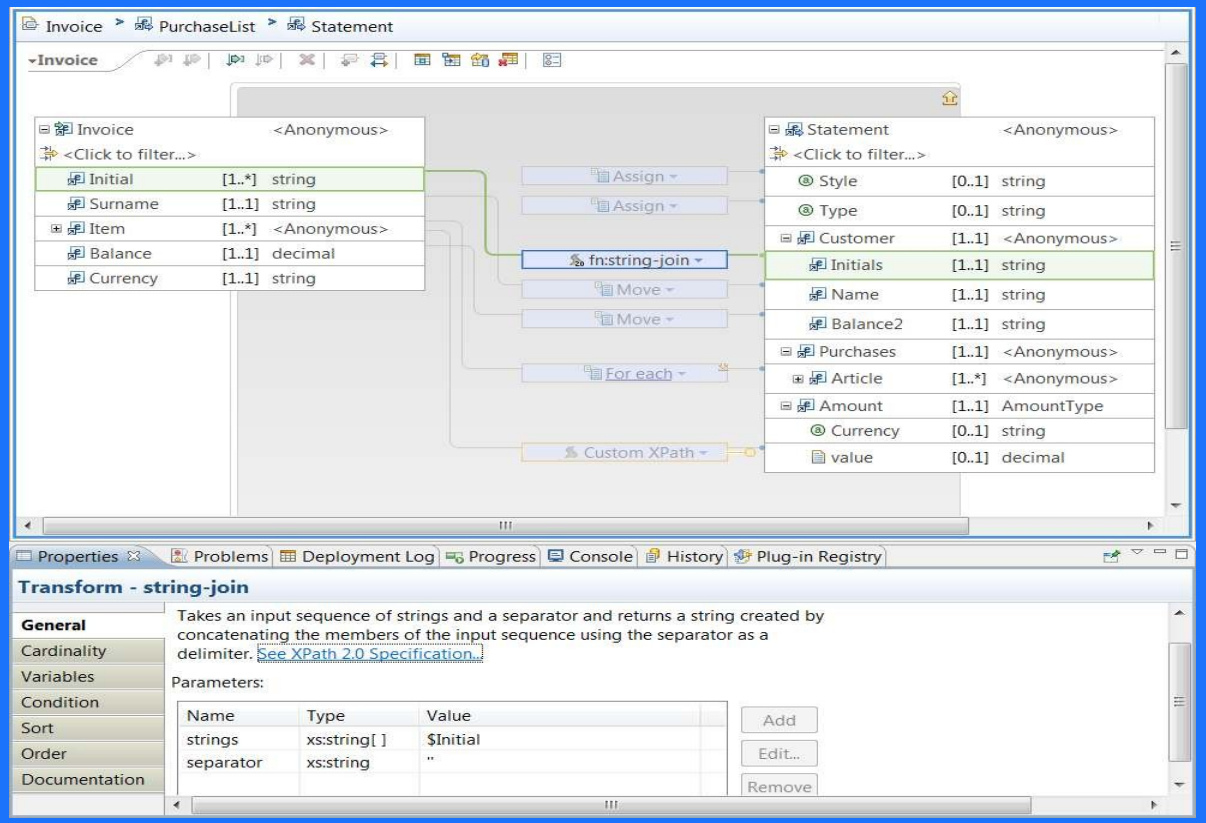
Transformation Options

- 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

- 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

Functionality

- Multi domain support
- Integrates with RDBMS
- Rich function library
  - Extensible with Java & ESQL

Ease of Use

- Gentle learning curve
- Graphical drag and drop
- Minimal ‘programming’ skills

Performance

- Compiles and runs in dedicated engine
- Benefits from JIT technology

Standards

- W3C XPath 2.0 expression language
- Structures modelled in XML Schema (XSD)

Maintenance

- Visual representation assists understanding

Portability

- Graphical Data Mapper used across many IBM products
  - Including DataPower MDM Server, RAD

Compute Node

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

Compute Node – at a glance

Functionality

- Full domain support
- Integrates with RDBMS
- Rich function library
  - Extensible with Java, .NET

Ease of Use

- Compact DSL (Domain Specific Language)
- Path navigation build directly into syntax

Performance

- Interpreter closely coupled to message tree
  - Excellent tree performance
- Easy to write poorly performing code though

Standards

- Built on SQL standard
- Proprietary extensions

Maintenance

- Built in source code debugger
- Supports development of reusable code libraries
- Deployed as source

Portability

- Runs on all platforms
- Unlikely to run anywhere else

JavaCompute

- 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

JDBCTestFlow_JavaCompute.java

```
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

- Full J2SE JVM
- Huge function library
  - JDK 7.0
  - 3rd party JARs
- Full domain support
- Excellent I/O capabilities

Ease of Use

- Standard Java + IIB API
- Navigation using XPath
- API-less transformation using standard JAXB
- Full JDT tools
- Beware of multi-threads

Performance

- Advanced JIT technology
- Thin IIB API around native IIB internals
- Easy to write poorly performing code though

Standards

- Java
  - Ubiquitous
  - JAXB, JDBC XA, …
- W3C
  - XPath 1.0

Maintenance

- Built in source code debugger
- Productivity tools
  - JUnit, JavaDoc, …

Portability

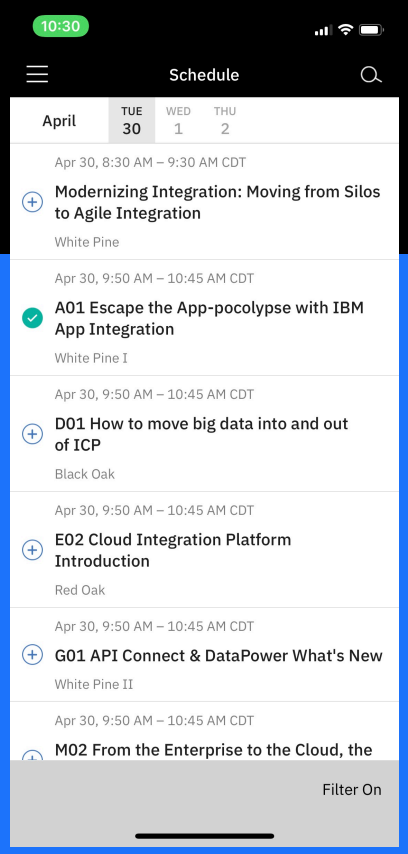
- Runs on all platforms
- JAXB transformation will run on other products and vendors

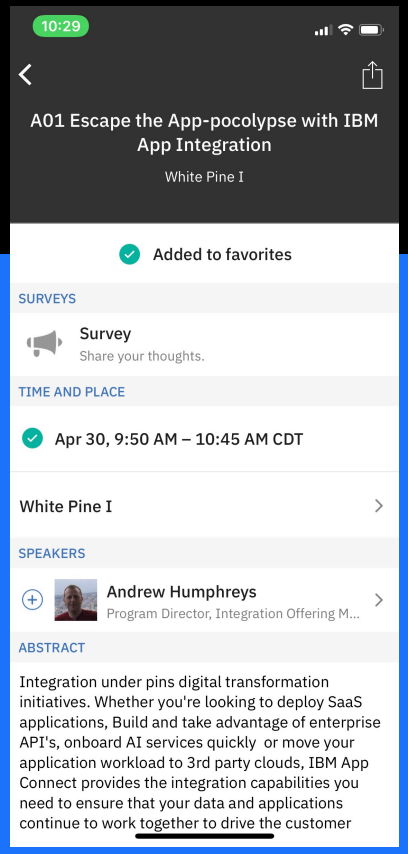
Summary

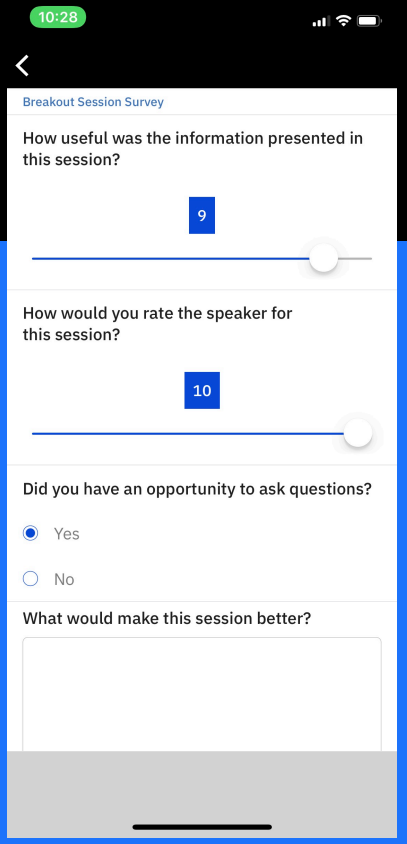
- Multiple areas in which recommendations can be applied
  - Design
  - Coding
  - Test
  - Deployment

- 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







Don’t forget to fill out the survey!

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

Thank You