



WebSphere ESB

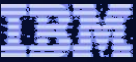
WebSphere Enterprise Service Bus V6.1 Update

Phil Coxhead
ISSW Worldwide Technology Practise
phil_coxhead@uk.ibm.com

4th March 2008

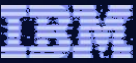
SOA on your terms and our expertise



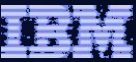


Agenda

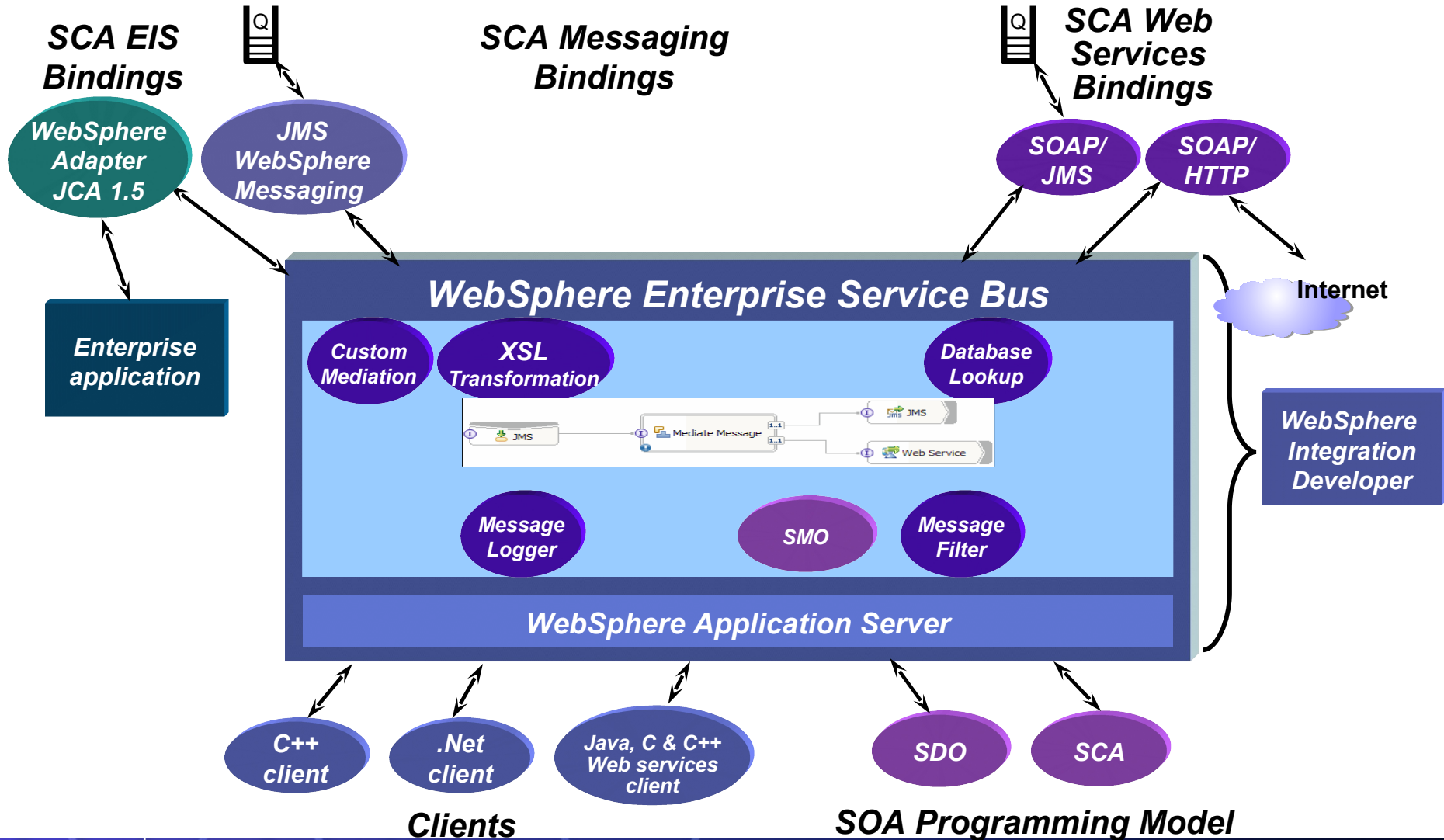
- Product Overview – a review
- Enhancements to existing features
- New features
- Enhancements to developer tooling

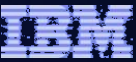


Review of V6.0.1 and V6.0.2 functionality

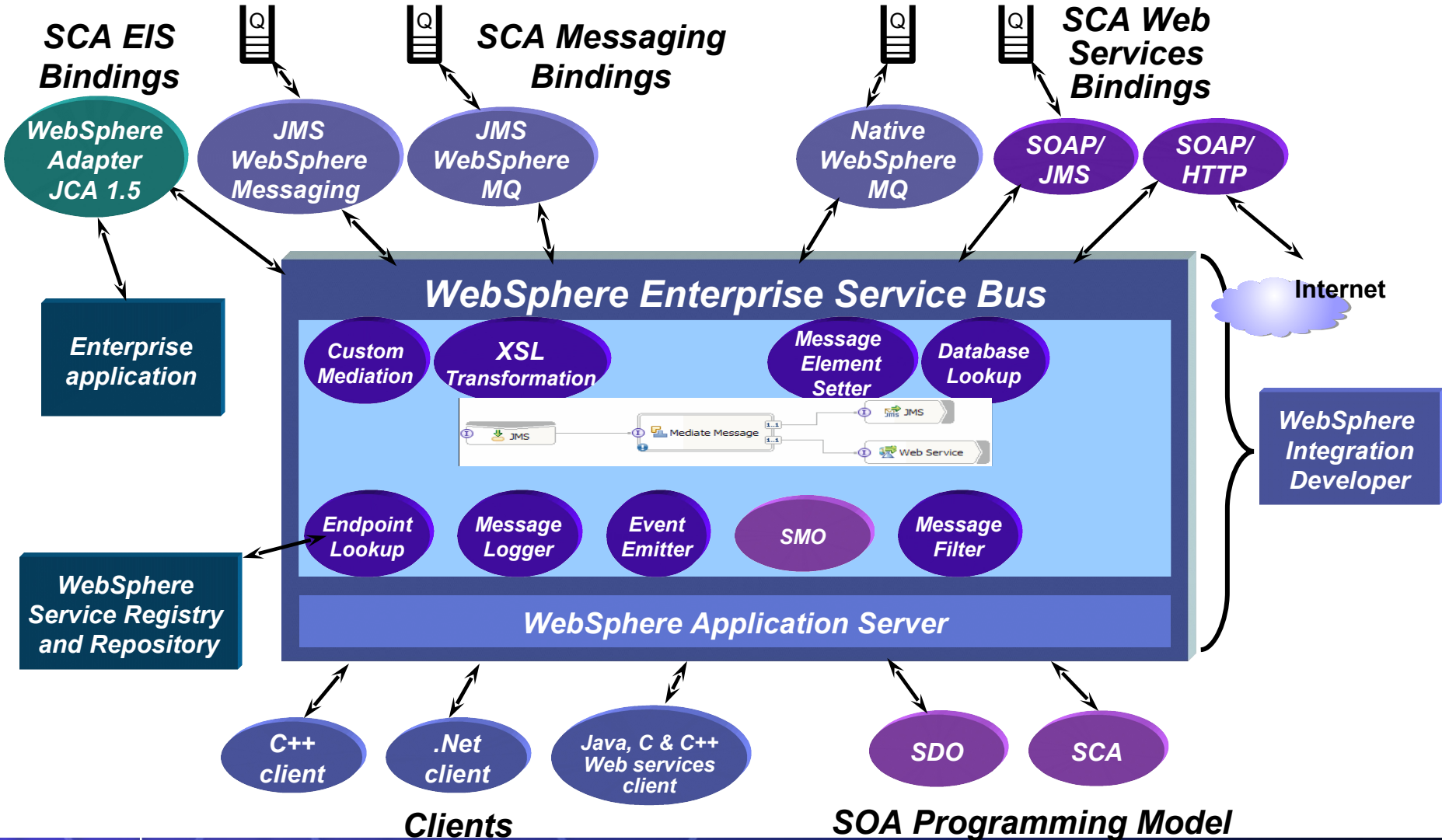


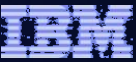
WebSphere ESB V6.0.1



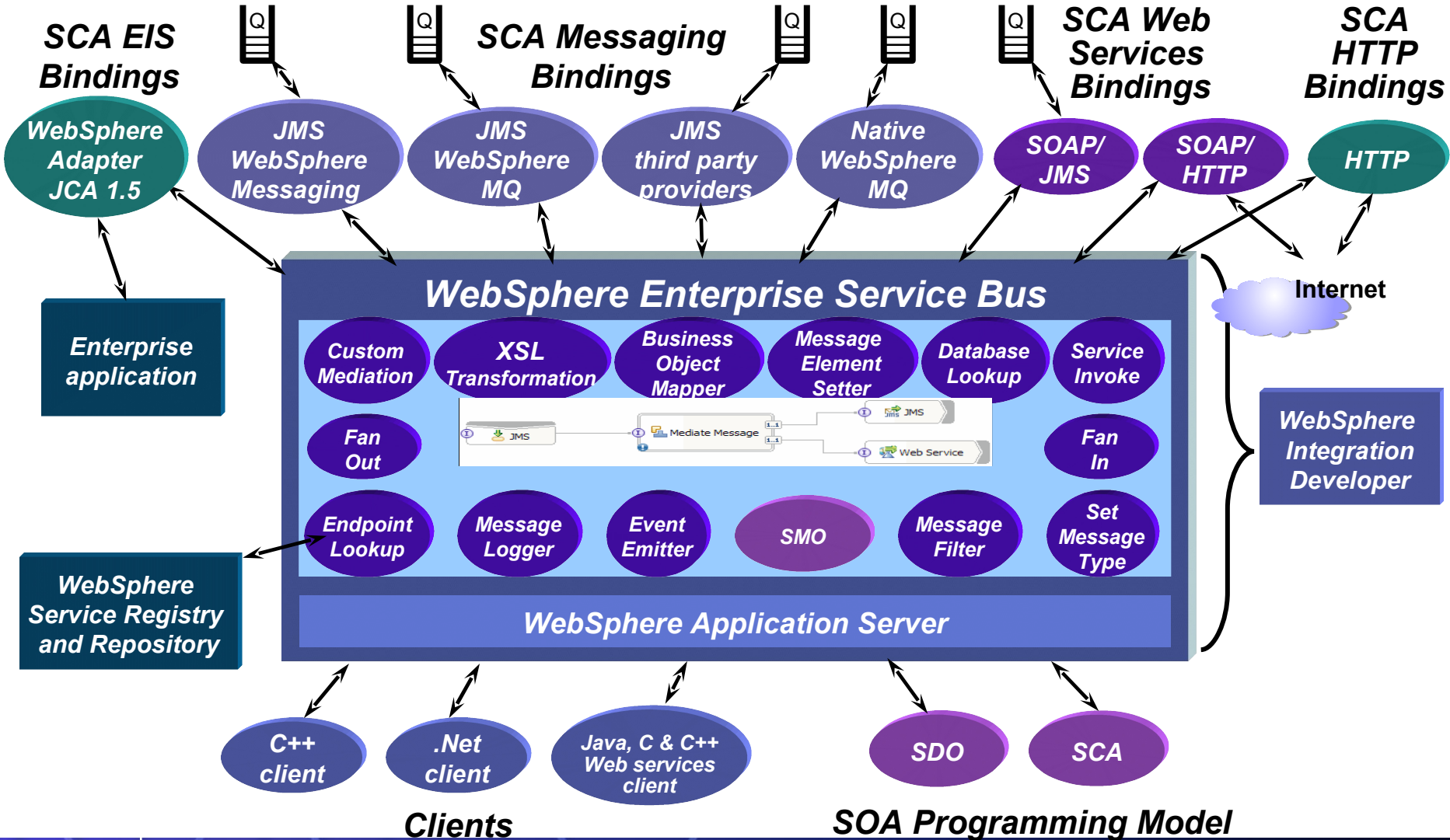


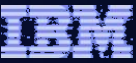
WebSphere ESB V6.0.2



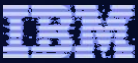


WebSphere ESB V6.1





Enhancements to existing features

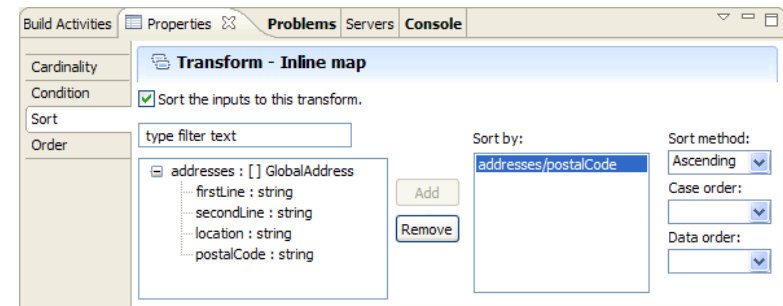
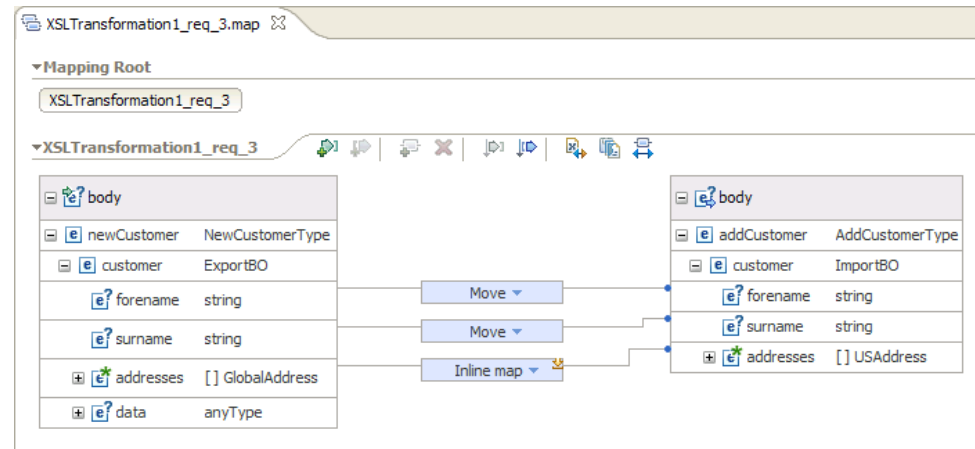


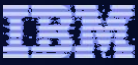
XSLT primitive enhancements



XSLTransformation1

- Mapping editor based on RAD V7
- Creation of XML test input
- Reuse of maps
 - across primitives
 - as sub-maps for handling complex elements
- Other mappings/refinements
 - Move – copy primitive type
 - Concat – concatenate multiple strings
 - Custom
 - XPath
 - Java – call static methods
 - XSLT – call external template
 - Inline map – nested map only usable in current map file
 - Substring – returns substring based on delimiter and index
- Arrays can be sorted and either iterated over or specify cardinality
- Conditional transformation





Custom mediation primitive enhancements

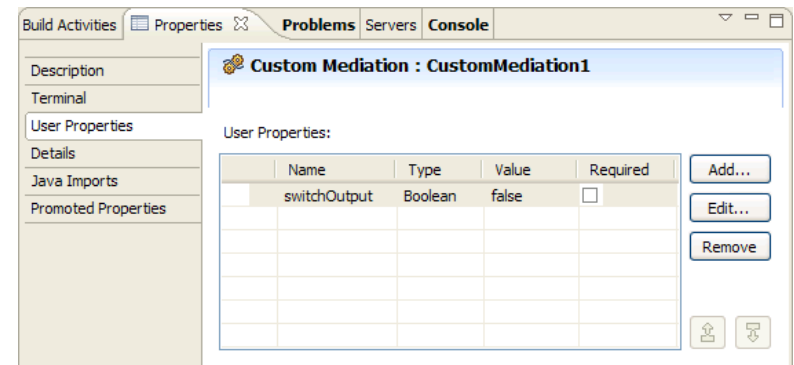
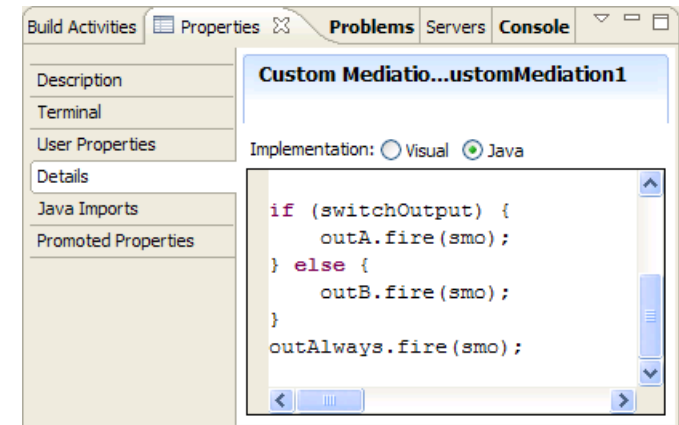
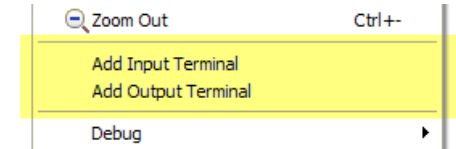


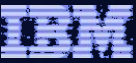
- Extends some of capabilities of user-defined primitive to custom mediation
 - One or more input terminals
 - Zero or more output terminals
 - Control over when output terminals fired
 - Ability to define (promotable) properties

- Separate panel for Java imports

- Ability to use SCA component as implementation removed – use new service invoke primitive

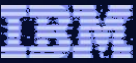
- Quick fix provided to migrate Java/Visual implementation to 6.1





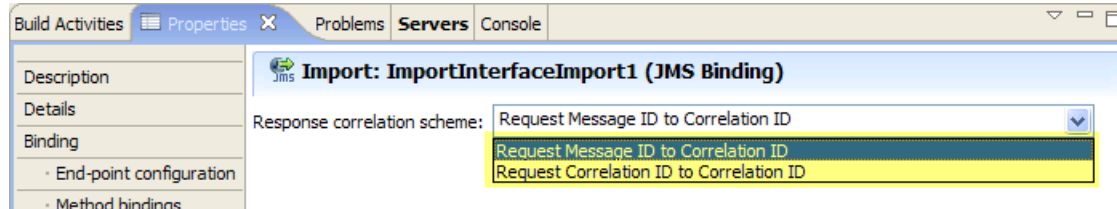
Message logger primitive enhancements

- Now logs to CommonDB by default for both single server and ND
- Script for generating resources now available for all supported databases
 - `<WESB_HOME>/dbscripts/EsbLoggerMediation`
- Schema name now overridden via `ESB_MESSAGE_LOGGER_QUALIFIER` environment variable

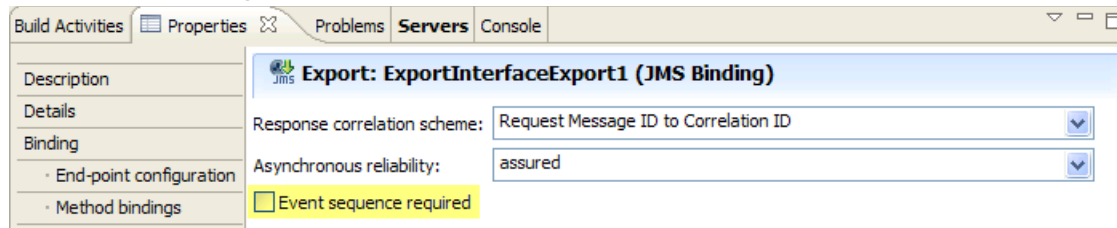


JMS binding enhancements

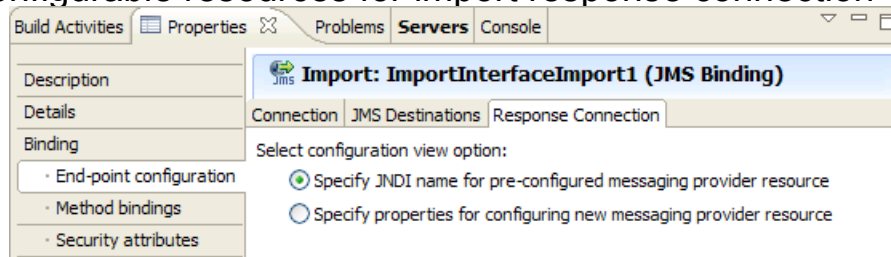
- JMS bindings brought in to line with WebSphere MQ JMS binding
 - Configurable correlation schemes for imports and exports

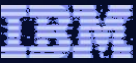


- Event sequencing for exports



- Configurable resources for import response connection





Administration of bindings

- Runtime administration previously only available for SCA and web service bindings
- Now also available for Generic JMS, JMS, MQ JMS and Native MQ (and HTTP) bindings
- Browse and configure destinations, connection factories, activation specs and listener ports
- AdminTask commands: (show|modify)SCA(Import|Export)(MQ|JMS)Binding

SCA modules

[SCA modules](#) > [WSTC](#) > **ExportInterfaceExport1**

The attributes of the selected JMS export binding.

Configuration

General Properties

Send Resources

Connection Factory JNDI Name

Send JMS Destination JNDI Name

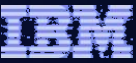
Receive Resources

Activation specification JNDI Name

Receive JMS Destination JNDI Name

Advanced Resources

Callback JMS Destination JNDI Name



Endpoint lookup primitive

- SSL repertoire configured on WSRR definition

- New match policy:
 - *Return all matching endpoints and set alternate routing targets*

- Policy sets new *AlternateTarget* element in SMO Header

General Properties

Connection type
Web service

* Registry URL
http://localhost:9080/WSRR.CoreSDO/services/WSRR.CoreSDOPort

Authentication alias
esbNode/wsrralias

SSL Configuration
wsrrsslconfig

Endpoint Lookup : EndpointLookup1

Name: ImportInterface

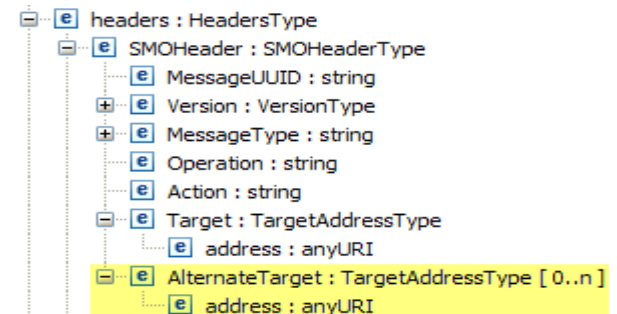
Namespace: http://WSTC/ImportInterface

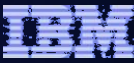
Version:

Registry Name: <Use default registry>

Match Policy:

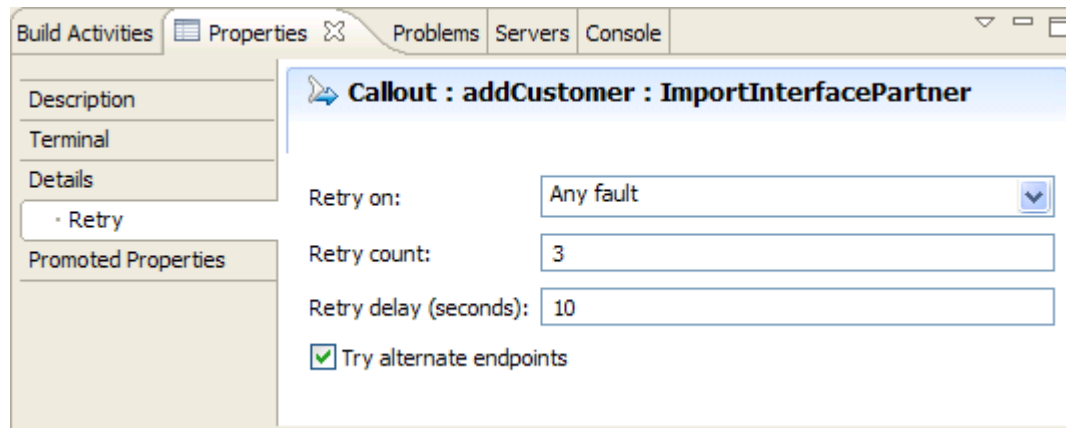
- Return first matching endpoint and set routing target
- Return all matching endpoints
- Return first matching endpoint and set routing target
- Return all matching endpoints and set alternate routing targets



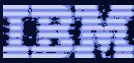


Retry on callout

- Enables retry attempts on failure to invoke referenced service

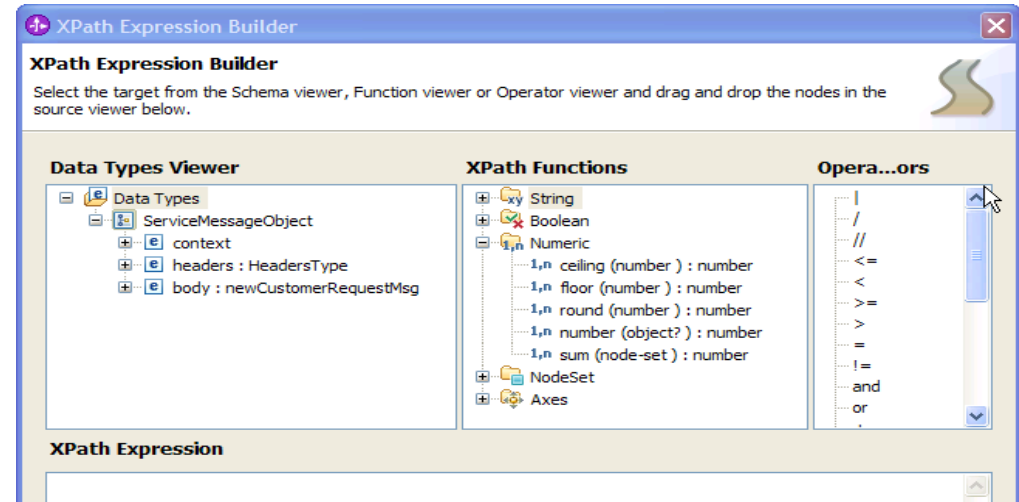


- **Retry on:** *Never* | *Any fault* | *Modeled fault* | *Unmodeled fault*
- **Retry count:** number of attempts after initial failure
- **Retry delay:** delay in seconds between attempts (from timeout for asynchronous requests)
- **Try alternate endpoints:** alternate endpoints from SMO header tried in order, wrapping around until retry count reached

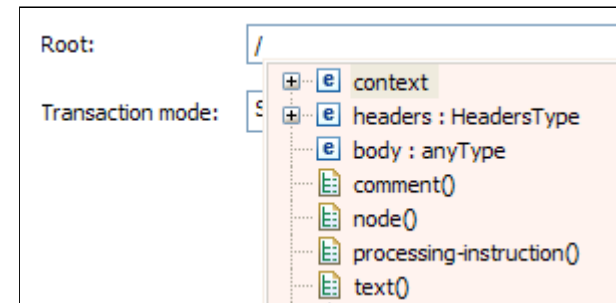


XPath tooling enhancements

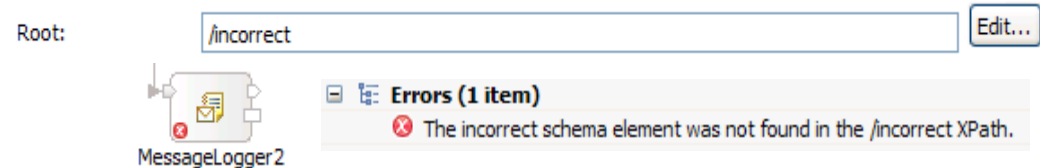
- Enhanced expression builder

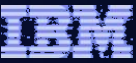


- Content Assist

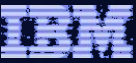


- Expression validation



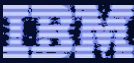


New runtime features



Deployment Environments

- Concept of deployment environments introduced to ease creation of ND topologies
- Three patterns supported out of the box:
 - Single cluster
 - Remote messaging
 - Remote messaging and remote support
- Clusters provide the following functions
 - Application deployment target
 - Messaging infrastructure
 - Support infrastructure
- Deployment environments can be created:
 - During installation
 - During profile creation
 - Via the administrative console
- Capability to view status and start and stop deployment environment



Service invoke primitive



ServiceInvoke1

- Invokes a service operation from within a request or response flow
- Request passed to *in* terminal; response flows from *out* or *fail*

Service Invoke : ServiceInvoke1

Reference name:

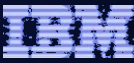
Operation name:

Use dynamic endpoint if set in the message header

Async timeout (seconds):

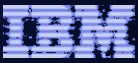
Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback.

- *Reference name* and *Operation name*: determine operation to invoke – specified on creation
- ***Use dynamic endpoint if set in the message header***
- ***Async timeout*** : time to wait in seconds for an asynchronous response (or -1 for infinite wait) before firing *timeout* terminal; treated like unmodeled fault with respect to retry
- *Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback*
- Same **retry options** as callout



Service invoke – invocation style

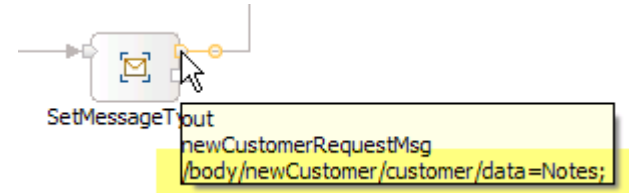
Mediation Flow Component Invocation Style	Target Preferred Interaction Style	One Way / Request-Response Operation	Force Sync	Invocation style used for Service Invoke
invoke (synchronous)	ANY	One Way	any	Async
		Request-Response	any	Sync
	SYNC	both	any	Sync
	ASYNC	both	any	Async
invokeAsync	ANY	One Way	any	Async
		Request-Response	any	Sync
	SYNC	both	any	Sync
	ASYNC	both	any	Async
invokeAsyncWithCallback	ANY	Request-Response	True	Async
			False	AsyncWithCallback
	SYNC	both	any	Sync
	ASYNC	One Way	any	Async
		Request-Response	True	Async
	False		AsyncWithCallback	



Set message type primitive



- Indicates down cast for weakly typed fields (e.g. derived types or xsd:any) in SMO to enable manipulation by tooling
- Message type augmented with additional type information propagated from *out* terminal

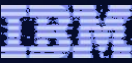


The screenshot shows the configuration window for the 'Set Message Type : SetMessageType1' activity. The 'Details' tab is selected, showing 'Message field refinements'.

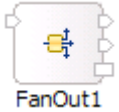
Weakly typed field	Actual field type
/body/newCustomer/customer/data	{http://WSTC}Notes

Below the table are two checkboxes: Reset message type and Validate. To the right of the table are buttons for 'Add...', 'Edit...', 'Remove', and two arrow icons for reordering.

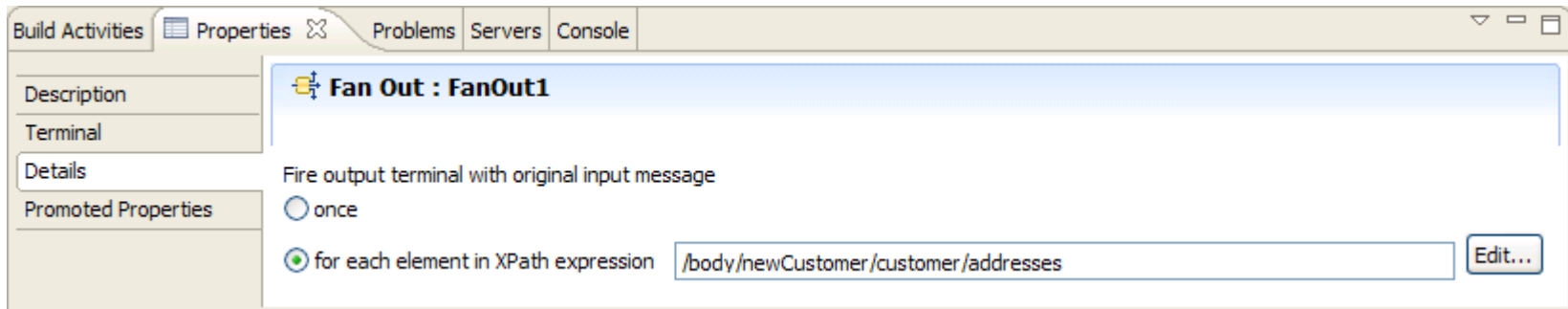
- Reset message type:** removes any existing type augmentations
- Validate:** performs runtime validation to ensure field is of specified type



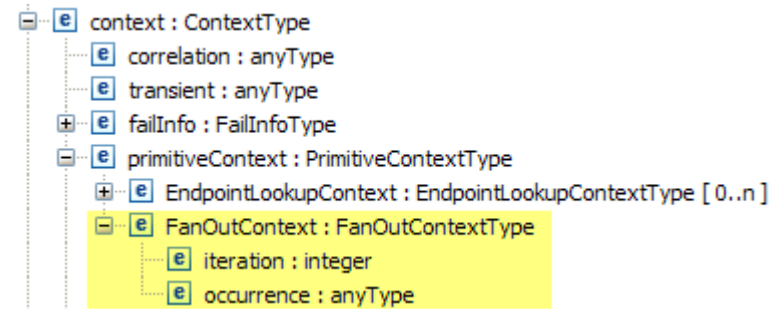
Fan out primitive

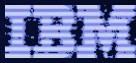


- Enables message to be processed by multiple paths or iterate over a repeated element
- No parallelism – synchronous processing of paths/elements



- *Fire output terminal with original input message*
 - *once*
 - *for each element in XPath expression*
 - FanOutContext contains *iteration* and *occurrence* (augmented type)
 - *noOccurrences* terminal fired if no elements found

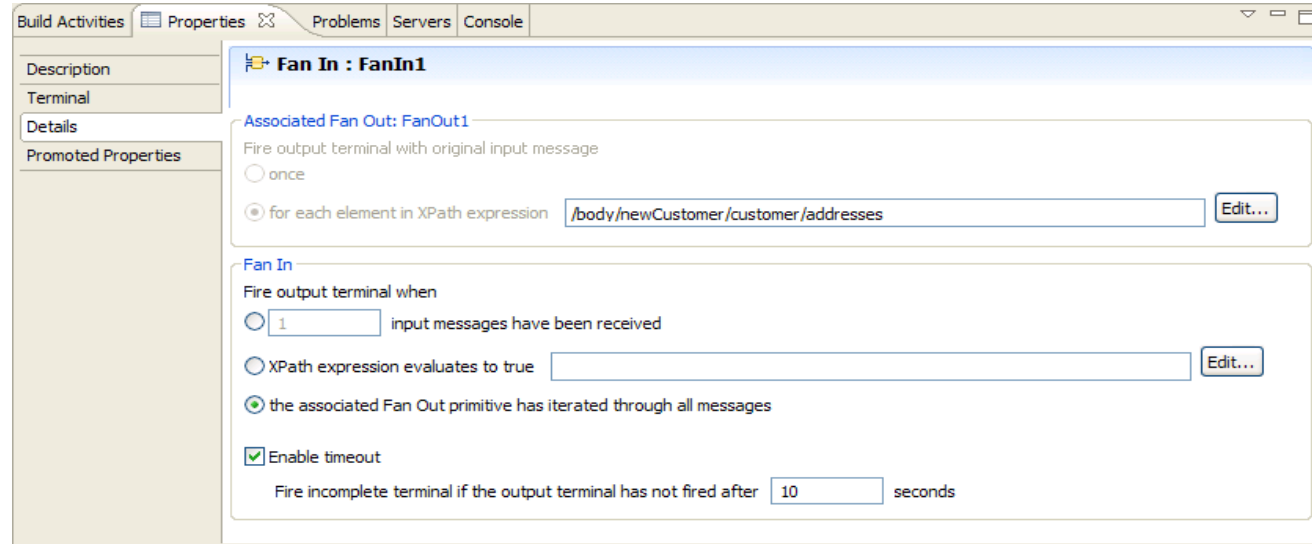




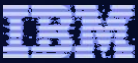
Fan in primitive



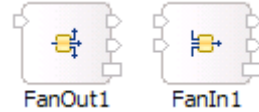
- Waits for messages from fan out until a completion criteria is met



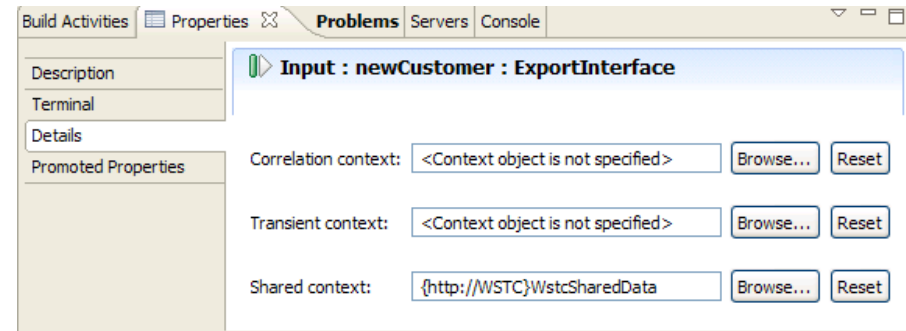
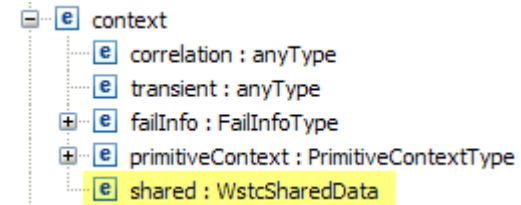
- Fire output terminal when*
 - X** input messages have been received*
 - XPath** expression evaluates to true*
 - the associated Fan Out primitive has iterated through all messages*
- Enable timeout*
 - Fire incomplete terminal if the output terminal has not fired after **Y** seconds*
- stop input terminal causes fan out/in to stop and fires incomplete terminal*

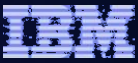


Shared context



- Fan in does not in itself do any aggregation – the *out* terminal is just fired with the last message received
- Each path from the fan out receives a deep copy of the message *except* for the shared context
- Shared context type defined on input node
- The path between the fan out and fan in should update this shared context
- The shared context is then propagated out from the fan in so can then be manipulated by subsequent primitives

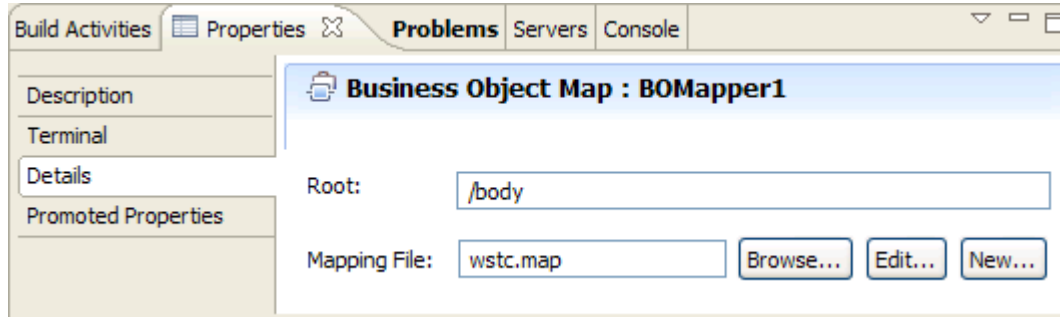




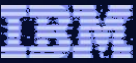
Business object map primitive



- Enables use of BO map for transformation of SMO

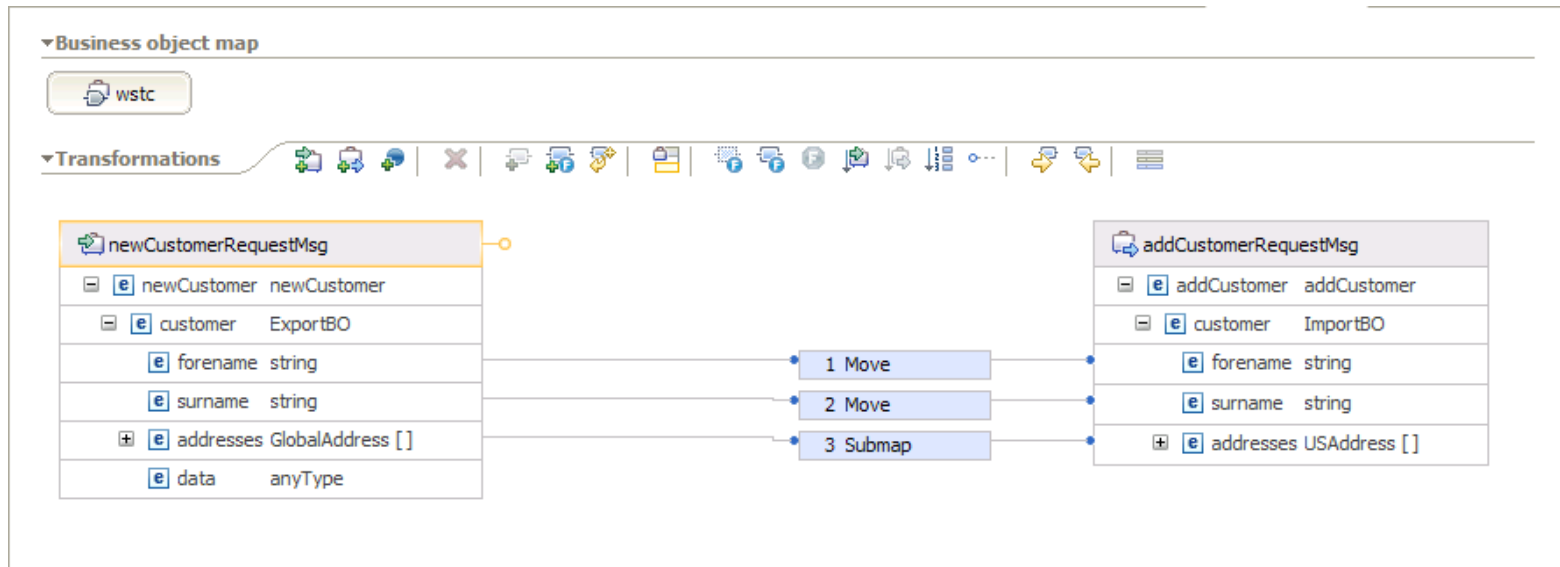


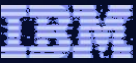
- **Root:** part of SMO to be transformed
 - /
 - /header
 - /context
 - /body
- Properties aren't editable – specified on creation of business object map



Business object map editor

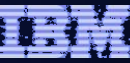
- Transformations – processed in designated order
 - Move, extract, join, assign, submap, relationship, relationship lookup, custom, custom callout, custom assign
- Variables available to store temporary values during mapping
- Option to map similar fields where names don't match
- Cardinality enables position in array to be specified
- Graph or table view





Business object map versus XSLT

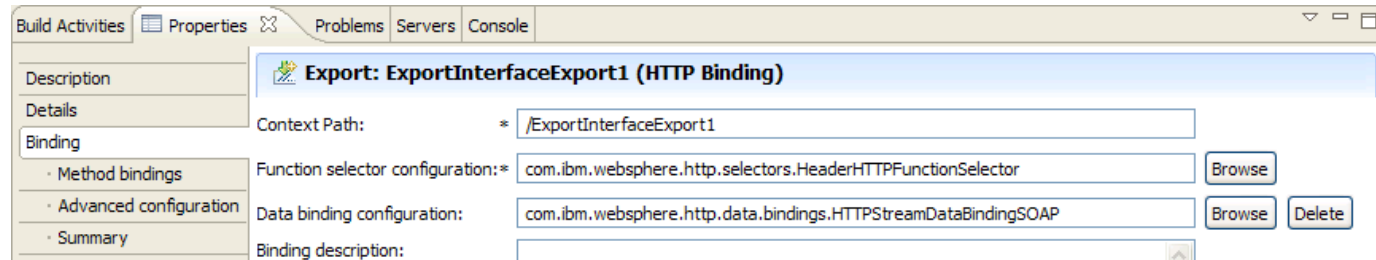
- Why use business object maps instead of XSLT?
 - Mapping requires maintaining a relationship
 - Change summary needs to be maintained in a business graph
 - Configure event settings to raise CEI events
 - Utilize existing investment in business object maps
 - Variables
 - Fuzzy mapping
- Performance
 - Business object map operates on service data object losing benefits of lazy parsing of raw data
 - Actual transformation may be faster with a business object map



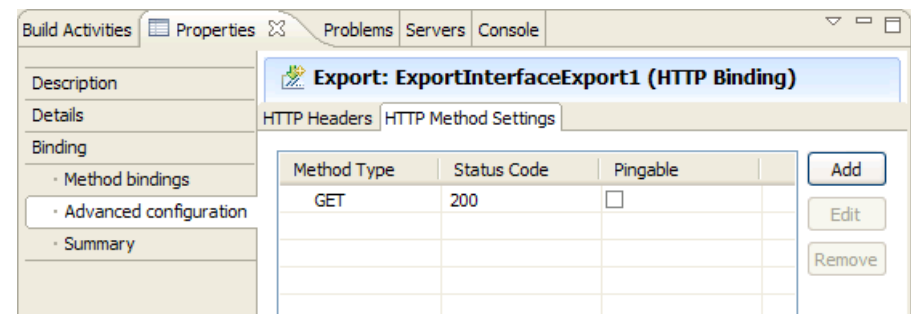
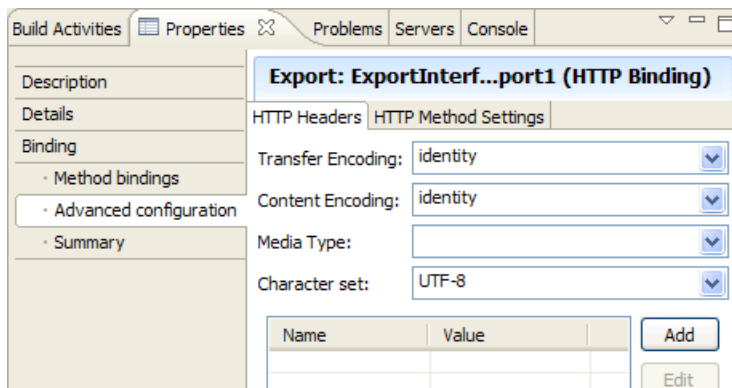
HTTP export binding

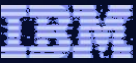


- Enables client connectivity via HTTP(S)
- Export context path, function selector and data binding specified on creation



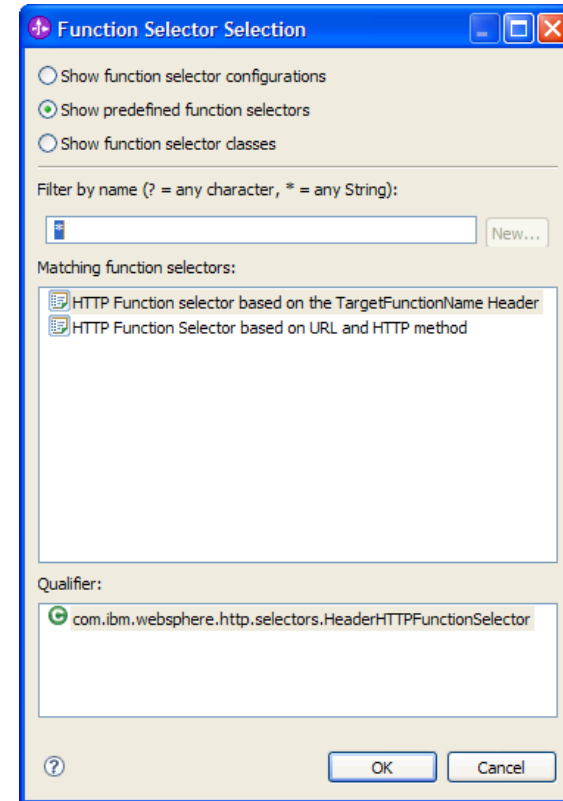
- Advanced configuration
 - HTTP headers for response
 - Supported HTTP methods and whether “pingable” (just status code returned)
- Methods bindings contain context path, native method(s) and enable other settings to be overridden on per operation basis



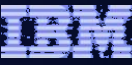


HTTP function selectors

- Two predefined function selectors
 - *TargetFunctionName* header
 - URL and HTTP method
 - Native method name =
 - HTTP Export Context Path
 - + Operation Context Path
 - + “@”
 - + HTTP method
 - For example:
 - /ExInterfaceExport1/newCustomer@GET



- Custom function selector
 - Extend *com.ibm.websphere.http.selectors.HTTPFunctionSelector*
 - Implement *String generateEISFunctionName(HTTPControl, HTTPHeaders, HTTPInputStream)*



HTTP import binding



- Enables invocation of service via HTTP(S)
- Endpoint URL and data binding specified on creation - option to generate TargetFunctionName

Build Activities | Properties | Problems | Servers | Console

Import: ImportInterfaceImport1 (HTTP Binding)

Description

Details

Binding

- Method bindings
- Advanced configuration
- Summary

Endpoint URL: *

Data binding configuration:

HTTP Version:

HTTP Method:

Binding description:

- HTTP version and method specified
- Advanced configuration
 - HTTP headers, proxy and SSL settings, and retry/timeout properties
- Method bindings allow all of above to be overridden on per operation basis

Import: ImportInter...ort1 (HTTP Binding)

HTTP Headers | HTTP Proxy | Security | Performance

Transfer Encod

Content Encod

Media Type:

Character set:

Import: ImportInterfa...mport1 (

Proxy Configuration

Host Name:

Port:

Non-Proxy Host Names:

Import: ImportInt...t1 (HTTP Binding)

HTTP Headers | HTTP Proxy | Security | Performance

SSL Authentication Alias:

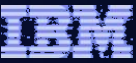
Basic Authentication Alias:

Import: ImportInt...t1 (HTTP Binding)

HTTP Headers | HTTP Proxy | Security | Performance

Read Timeout:

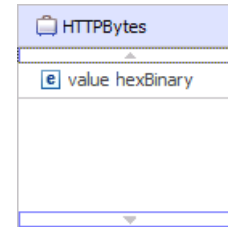
Number of Retries:



HTTP data bindings

Supplied HTTP data bindings

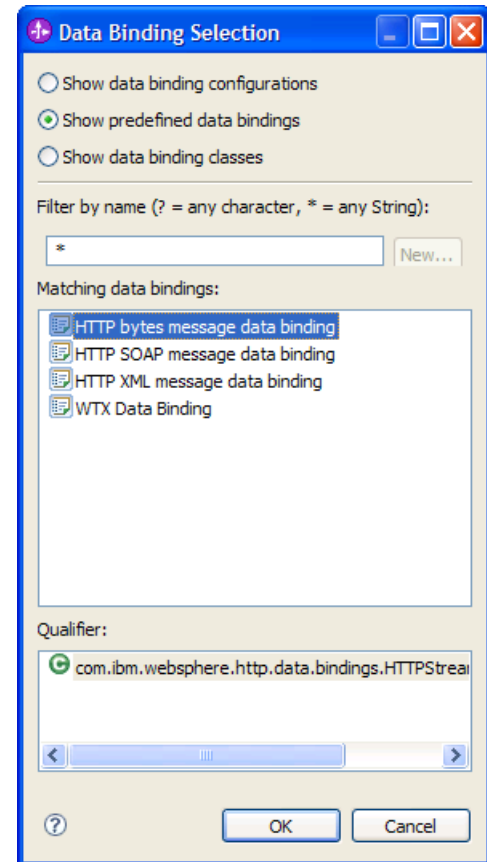
- HTTP bytes message
 - Request represented as XSD hexbinary
 - Add dependency for HTTPBytes business object
 - Operations must take/return HTTPBytes BO
- HTTP SOAP message
- HTTP XML message



Predefined Resources

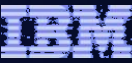
Select the resources to import into this module.

- WS-Addressing Schema Files
- Microsoft ADO.NET DataSet schema file
- Schema for simple JMS Data Bindings
- Schema for predefined HTTP bytes data binding



Custom data binding

- Implement `com.ibm.websphere.http.data.bindings.HTTPStreamDataBinding`
 - `get/setDataObject`
 - `setBusinessException/isBusinessException`
 - `get/setControlParameters`
 - `get/setHeaders`
 - `convertFromNativeData(HTTPInputStream)`
 - `convertToNativeData()`
 - `write(HTTPOutputStream)`

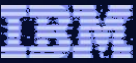


Generic JMS binding



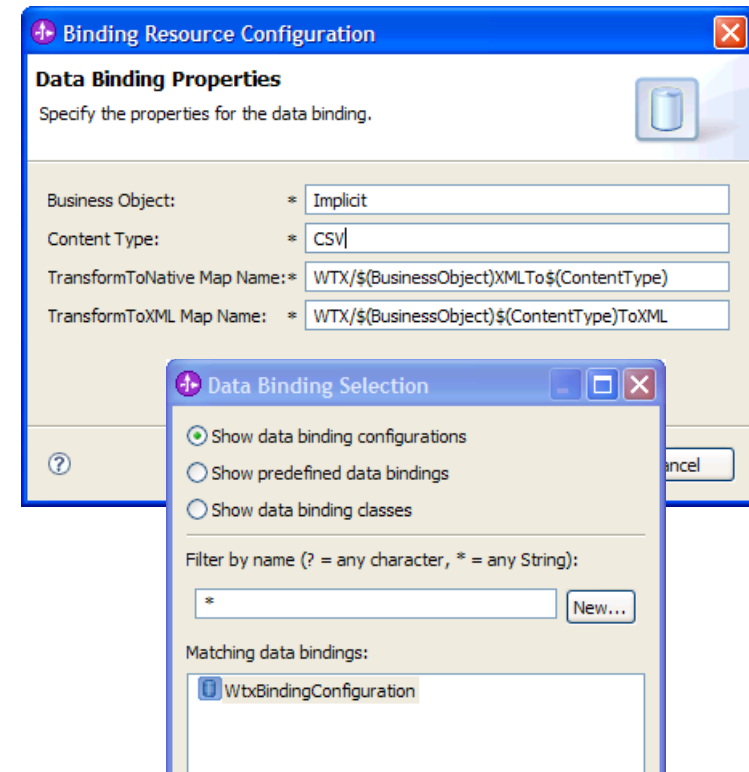
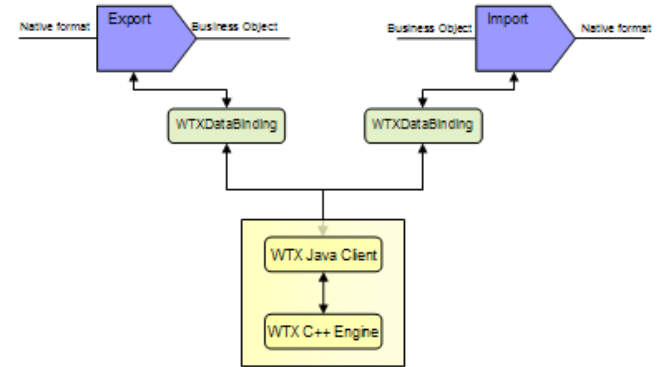
- Provides same capabilities for generic JMS providers as WebSphere MQ JMS and JMS bindings
- Provider be JMS 1.1 compliant and implement Application Server Facilities
- Resources can be preconfigured via application server's Generic JMS provider support or generated (name provider must already exist)

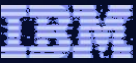
The screenshot displays the configuration interface for a Generic JMS Binding. The main window title is "Import: ImportInterfaceImport1 (Generic JMS Binding)". The left sidebar shows a tree view with "Binding" selected, containing sub-items: "End-point configuration", "Method bindings", "Security attributes", "Message configuration", and "Summary". The main content area is divided into "Request" and "Response" tabs, with "Request" active. Under "Request", there is a section for "Connection Factory Properties" with two radio button options: "Specify JNDI name for pre-configured messaging provider resource" (selected) and "Specify properties for configuring new messaging provider resource". Below this is a text field for "JNDI Lookup Name:" containing the value "jms/cf". A similar section for "Send Destination Properties" is visible below, also with the "Specify JNDI name" option selected and a "JNDI Lookup Name:" field containing "jms/send".



WTX data binding

- WebSphere Transformation Extender provides universal data transformation and validation
- WTX data binding applicable to imports and exports with messaging, EIS and HTTP transport bindings
- Uses JNI to invoke WTX (8.2) transformation engine running in-process
- Data binding configuration defined via *New > Binding Resource Configuration* and then selected on creation of import/export binding
- Default is to find WTX map file in SCA module based on business object name and configured content type e.g. WTX/CustomerCSVToXML.mmc

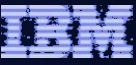




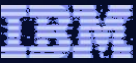
Runtime Business Object Validation

- Execution-time validation of request and response data against schema
 - Applies to all operations when enabled
 - Specified as an SCA Qualifier on an interface
- Reporting options
 - Validation errors are logged and running application continues
 - Exceptions thrown, potentially application dies

The screenshot displays the configuration interface for a component named "PIPProcess (Process)". The left-hand pane shows a tree view with "Interfaces" expanded to show "EncryptionInterface" (with an "encrypt" operation) and "References". The main right-hand pane has tabs for "Details", "Qualifiers", and "Event Monitor". The "Qualifiers" tab is active, showing a list of "Quality of Service (QoS) Qualifiers" including "Data validation" and "Join transaction". The "Data validation" qualifier is selected, and its properties are shown in a separate pane below. This pane is titled "Properties of Qualifier Data validation" and includes a dropdown menu for "Action for errors:" with the following options: "Log error and continue" (selected), "Log error and continue", and "Throw exception".

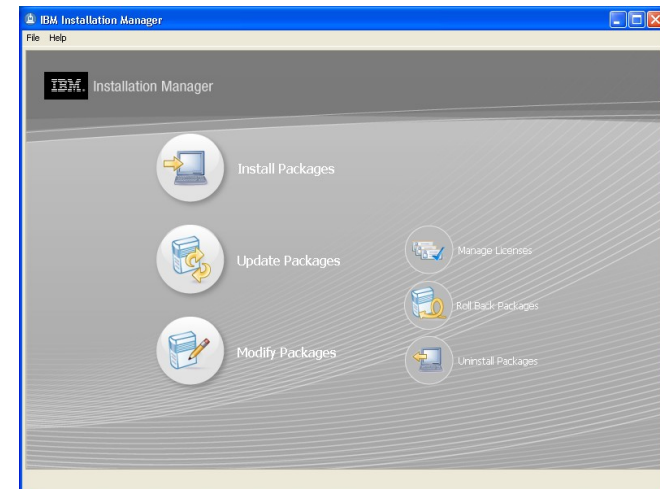


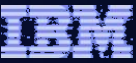
Enhancements to developer tooling



Installation Manager

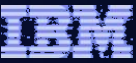
- **Install the whole product stack simultaneously** when using multiple IM-based products (e.g., RAD, WID, MB toolkit, Monitor toolkit)
- Install product updates during product installation
- Go from product install to latest level and skip the updates in between (upgrade from 1.0->1.5 without installing the interim levels)
- **Combine products into different configurations** and still share plugins if the same versions are installed (e.g., one configuration with WID/RAD, another with just WID)
- **Service both IDE and UTE through the same mechanism**
- Uninstall older versions of plugins when updates are applied (avoid disk space bloat over time)





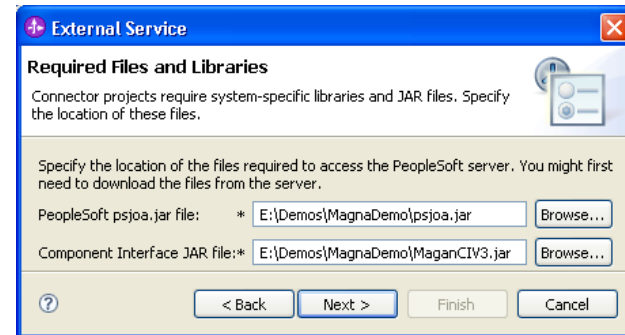
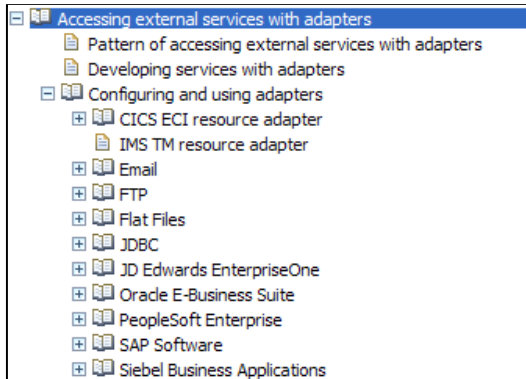
Major Categories of Improvements

- Full (tools and runtime) support for additional XSD/WSDL constructs
 - xsd:choice, xsd:any family, xsd:union
- Support for popular industrial schemas such as HL7 (Health Care), ACORD (Insurance), OAGIS (B2B), and others
 - Some documented fixes requires, schemas have bugs too!
- Numerous fixes
 - i.e. Problem: xsd:attribute cannot have the same name as xsd:element
- Validation
 - Consistent WSDL and XSD validation in WID and WPS
 - Faster and better WSDL validator from RAD
 - New runtime message validation
 - Validation for: xs:ID, xs:IDRef, xs:NMTOKEN, xs:Key, xs:KeyRef

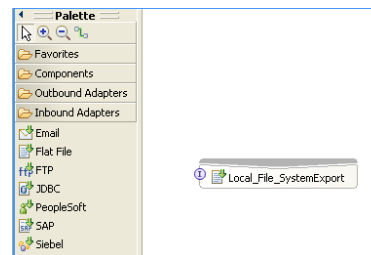


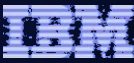
J2C Adapters – Key New Features

- Documentation included in Info Center
- Adapters Wizards asks for the adapter specific jars adds them to your adapter projects (i.e. People Soft jars)



- Redesigned Adapter Wizards – greatly improved usability
- Drag and drop adapter from the Assembly Editor palate
 - Adapter RAR file is installed automatically!
 - Adapter Wizard is launched





Integrated Test Client Enhancements

Events

- Invoke (mainProcess:applyOnline)
- Invoke (mainProcess:applyOnline)
 - Invoke started
 - Invoke (mainProcess:applyOnline)
 - Request (mainProcess --> LoanLimits:checkCredit)
 - Request (LoanLimits --> CreditCheck:checkCredit)
 - Response (LoanLimits <-- CreditCheck:checkCredit)
 - Response (mainProcess <-- LoanLimits:checkCredit)
 - Request (mainProcess --> FollowUpDeclinedApp:FollowUpDeclinedApp)
 - Return (mainProcess:applyOnline)
 - Invoke returned

General Properties

Detailed Properties

Configuration: Default Module Test

Module: LoanApplicationModule

Component: mainProcess

Interface: mainProcessInterface

Operation: applyOnline

Initial request meters

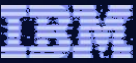
Name	Type	Value
applicationInfo	ApplicationBO	✓
applicant	ApplicantBO	✓
name	string	Paul Pacholski
email	string	✓
taxPaye	string	✓
loanamount	double	✓ \$10000.00

Maximize editor

10000.00 is not of type double

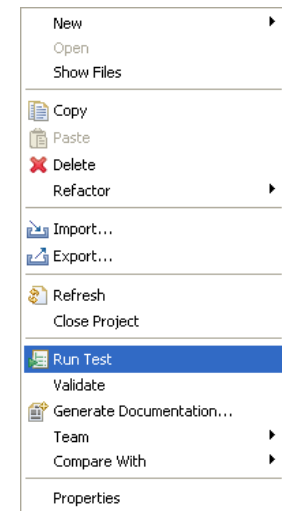
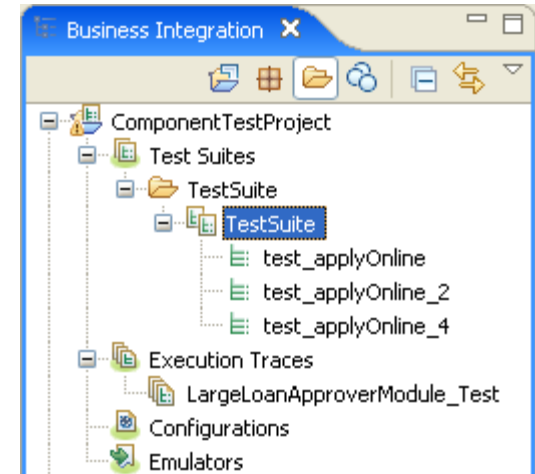
- Buttons repositioned to where they are used
- New input UI
 - Icons denoting structure types and input state
 - Parameter validation

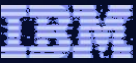
- Status area and error columns simplify error identification
- Maximize Editor button for easier data entry
- Multi-line data entry



Support for Application Testing

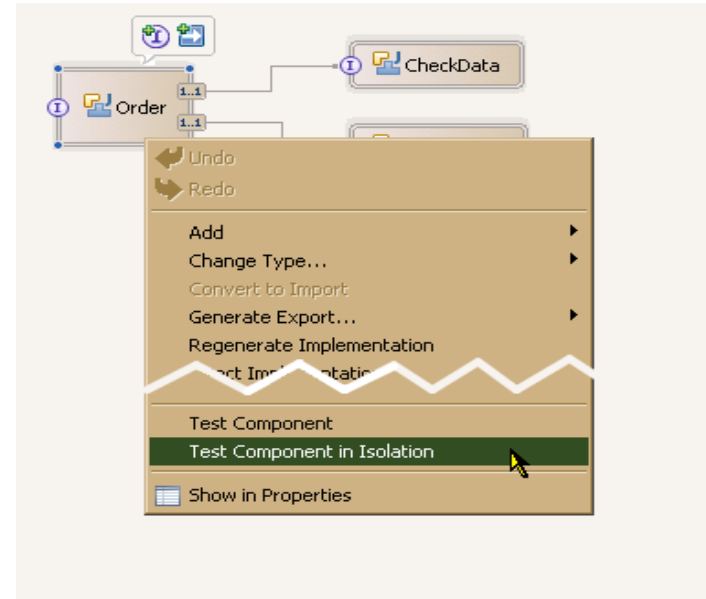
- Users can automate testing of WID applications
- New Component Test Projects
 - Test Projects contain Test Suites
 - Test Suites contain Test Cases
 - Test Cases can be authored:
 - Manually
 - From Test Client invocation trace
- Test execution is triggered from a Component Test Project
 - Test are deployed to server and executed
 - Results appear within the Integrated Test Client
 - Can be invoked from an ant script
- Test Variations
- Test Scenarios

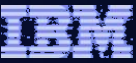




Test component in isolation

- 6.0.x: Test Component meant test selected component, emulate everything else
 - Not what user expects
- 6.1: Test Component means test module, but start with selected component
 - What users expect
- 6.1: Test Component in Isolation: test selected component, emulate everything else
 - Test Component behavior in 6.0.x was useful in some cases, so now it is named for what it does



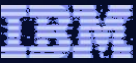


Choice support

```
<xsd:complexType name="BookInfo">
  <xsd:choice>
    <xsd:element name="ISBN" type="xsd:string" />
    <xsd:element name="Book" type="bons0:BookItem" />
  </xsd:choice>
</xsd:complexType>
```

- 6.0.x: `<xs:choice>` was treated as an `<xs:sequence>`
 - Required user to unset all but one element
 - User needed to be familiar with the schema

Name	Type	Value
<input type="checkbox"/> input1	BookInfo	
ISBN	string	<unset>
<input type="checkbox"/> Book	BookItem	
id	string	123
author	string	Bob
title	string	My Story

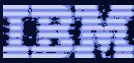


Choice support

- 6.1: Select a choice element
 - Unselected elements are unset
 - Click the Choice cell to change selected element
- xs:union treated similarly

Name	Type	Value
input1	BookInfo	✓
Choice	choice	68
Book	BookItem	✓
id	string	✓ 123
author	string	✓ Bob
title	string	✓ My Story

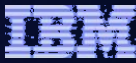
Name	Type	Value
input1	BookInfo	✓
Book BookItem	BookItem	68
ISBN	string	✓
Book BookItem	BookItem	✓ 123
au	string	✓ Bob
titl	string	✓ My Story



Testing thru the edges

- 6.0.x: Testing export == testing component wired to export
- 6.1: Test client looks at export binding
 - SCA: Equivalent to calling from an import in another module
 - Web Service: Client sends soap message to the address in the WSDL binding

Name	Type	Value
Envelope	Envelope	✓
Header	Header	✓
+ any	anyType[]	✗
anyAttribute	anySimpleType[]	✗✓
Body	Body	✓
any	anyType[]	✗
SimpleChoice	SimpleChoice	✓
input1	BookInfo	✓
Choice	choice	✗
ISBN	string	✓
anyAttribute	anySimpleType[]	✗✓
any	anyType[]	✗
anyAttribute	anySimpleType[]	✗✓



Authoring Test Cases

1. Collection of tests in a test case
2. A test case may include a series of operations
 - i.e. An interaction with state machine
3. Each test has separate input data
4. Each test has expected return value
 - Used to determine if the test failed or succeeded
 - Has expression builder

Test Cases

Test Cases

- test_applyOnline (1)
 - Invoke mainProcess:applyOnline
- test_applyOnline_4
 - Invoke mainProcess:applyOnline
- test_applyOnline_2
 - Invoke mainProcess:applyOnline
- test_applyOnline_3 (2)
 - Invoke mainProcess:applyOnline
 - Invoke CompleteTheLoan:CompleteTheLoan
- test_applyOnline_5
 - Invoke mainProcess:applyOnline

General Information

Description: Invoke: applyOnline()

Detailed Properties

Module: LoanApplicationModule
 Component: mainProcess
 Interface: mainProcessInterface
 Operation: applyOnline

Asynchronous mode

Request | Response | Exception

applicationInformation: ApplicationBO applicationInfor

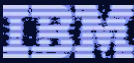
Overview | Test Cases | Configurations

Properties | Problems | Servers | Console | Test Data Table | Build Activities

Imports | Default

TestSuite : test_applyOnline

Name	Type	In	Expected
Invoke mainProcess:apply			
applicationInformation	ApplicationBO	✓	
applicant	ApplicantBO	✓ ...	
name	string	✓ Paul Pacholski	
email	string	✓ pacholsk@ca.ibm.com	
taxPayerId	string	✓ EXW456	
loanamount	double	✓ 100000	
Verify applyOnline			
response	string		Approved (4)



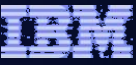
Executing Test Cases

The screenshot displays the IBM WebSphere ESB Test Client interface. On the left, a project tree shows a 'Component Test Project' with a 'Test Suite' and 'Test' folder. A context menu is open over the 'Test' folder, with 'Run Test' highlighted (2). The 'Events' window in the center shows a tree view of the test execution: 'Run Test (TestSuite) [Running]' (3), 'Test Suite (TestSuite) [Running]', and several test cases and variations with their statuses (4). On the right, the 'General Properties' and 'Detailed Properties' panels show the 'Configuration' as 'TestSuite', 'Verdict' as 'Running' (5), and a progress bar for 'Total: 4/5' with a green bar indicating 4/5 passed. Below the events window, a table shows the status of various servers (1):

Server	Status	State
WebSphere Process Server v6.1	Started	Synchronized
LargeLoanApproverModuleApp	Started	
LoanApplicationModuleApp	Started	
ComponentTestProjectApp	Started	

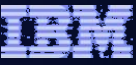
1. Component Test Project is automatically deployed to the server together with the module under test
2. Launching the Test opens Test Client

1. Toolbar to launch the test
2. Detailed invocation trace is optionally available
3. Test execution totals are shown for a Test Suite

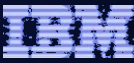


Summary

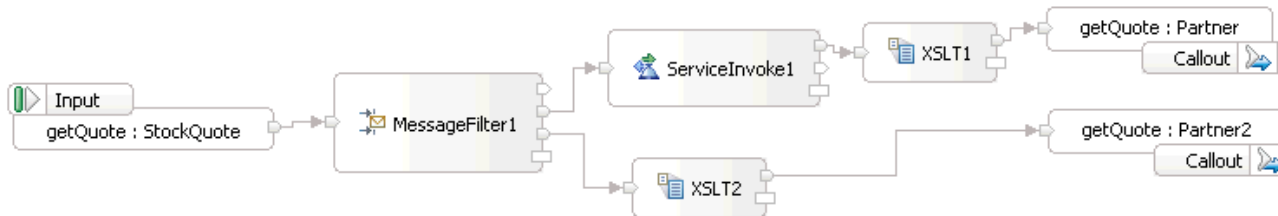
- Product Overview – a review
- Enhancements to existing features
- New runtime features
- Enhancements to developer tooling



Backup



Service Invoke – Parallel Threads Example



At run time the flow on thread one might be as follows:

1. The first matching output terminal of MessageFilter1 is fired.
2. The input terminal of ServiceInvoke1 receives the input message, and ServiceInvoke1 performs an `asyncWithCallback` invocation. The call does not wait for a response and returns immediately.
3. The thread traces back up the wiring path to the MessageFilter1 mediation primitive.
4. The second matching output terminal of MessageFilter1 is fired.
5. The input terminal of XSLT2 is reached, and the transformation occurs.
6. The response to the invocation made by ServiceInvocation1 is received. The flow engine handles this and resumes the processing in ServiceInvoke1 on a new thread.

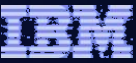
From now on, the threads can process in parallel.

The following actions occur on thread one:

1. The output terminal of XSLT2 is fired.
2. The Partner2 callout is invoked. This results in the invocation of the reference, and whatever the reference is wired to. For example, the sending of a SOAP message from a Web services import.
3. Because this is the end of the wiring path, the thread tracks back up the wire to the MessageFilter1 primitive.
4. There are no more output terminals to fire. Therefore, MessageFilter1 has completed and the thread tracks the wire back out of the input terminal to the previous mediation primitive, which in this case is the Input node.
5. At this point, thread one has completed.

The following actions occur on thread two:

1. The input terminal of XSLT1 is reached, and the transformation occurs.
2. The output terminal of XSLT1 is fired.
3. The Partner callout is invoked. This results in the invocation of the reference, and whatever the reference is wired to. For example, the sending of a SOAP message from a Web services import.
4. Because this is the end of the wiring path, the thread tracks back up the wire to the ServiceInvoke1 primitive.
5. At this point, thread two has completed.



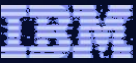
Service invoke for message augmentation

The screenshot displays the IBM WebSphere ESB Studio interface. At the top, the 'StoreMediation - Assembly Diagram' window shows a mediation flow starting with an 'Input' component (labeled 'submitOrder : Ord...') that branches into two paths: one leading to 'ShippingJava' and another to 'InventoryPartner'. A purple arrow points from the 'InventoryPartner' component in this diagram to the 'CheckInventory' component in the main mediation flow below.

The main mediation flow consists of a sequence of components: 'Input' (submitOrder : Ord...), 'Order2Inventory', 'CheckInventory', 'Inventory2Ship', and 'Callout' (shipOrder : Shippi...). A purple arrow points from the 'CheckInventory' component to the 'Service Invoke : CheckInventory' configuration window.

The 'Service Invoke : CheckInventory' configuration window shows the following settings:

- Reference name:
- Operation name:
- Use dynamic endpoint if set in the message header
- Async timeout (seconds):
- Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback.



Fan-out/fan-in for iteration



Build Activities | Properties | Problems | Servers

Fan Out : StartIteration

Fan Out

Fire output terminal with original input message

once

for each element in XPath expression [Edit...](#)

Associated Fan In: Enditeration

Fire output terminal when

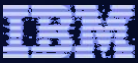
1 input messages have been received

XPath expression evaluates to true [Edit...](#)

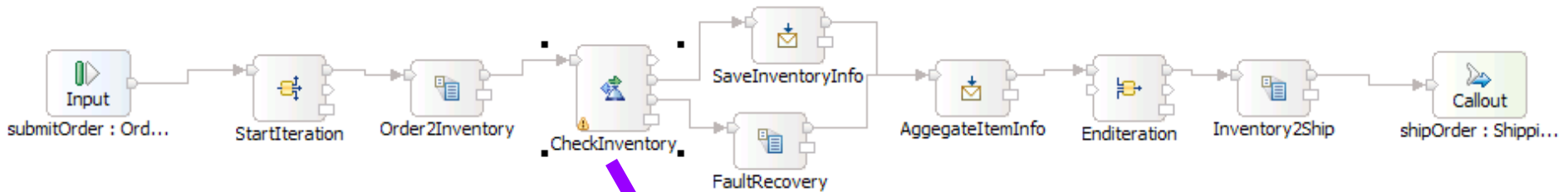
the associated Fan Out primitive has iterated through all messages

Enable timeout

Fire incomplete terminal if the output terminal has not fired after seconds



Retry for failure handling



Build Activities | Properties | **Problems** | Servers

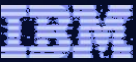
Service Invoke : CheckInventory

Retry on: Modeled fault

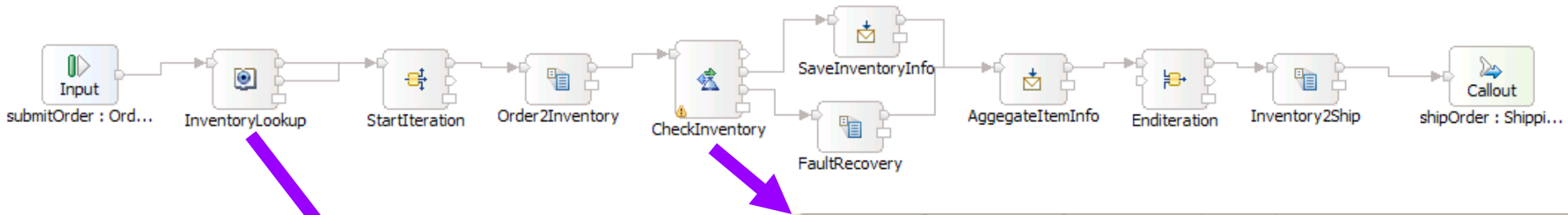
Retry count: 3

Retry delay (seconds): 0

Try alternate endpoints



Alternate endpoints for failure handling



Endpoint Lookup : InventoryLookup

Name:

Namespace:

Version:

Registry Name:

Match Policy:

Service Invoke : CheckInventory

Reference name:

Operation name:

Use dynamic endpoint if set in the message header

Async timeout (seconds):

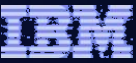
Service Invoke : CheckInventory

Retry on:

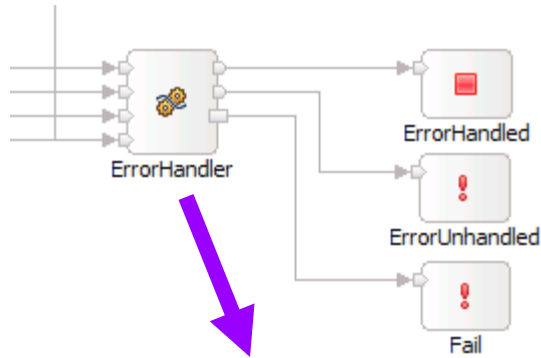
Retry count:

Retry delay (seconds):

Try alternate endpoints



Custom mediation for fault handling



Build Activities | Properties | Problems | Servers

Custom Mediation : ErrorHandler

Description

Terminal

User Properties

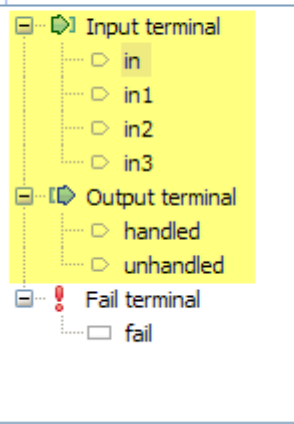
Details

Java Imports

Promoted Properties

Terminal name: in

Message type: {http://StoreLib/Ordering}submitOrderRequestM:



Build Activities | Properties | Problems | Servers

Custom Mediation : ErrorHandler

Description

Terminal

User Properties

Details

Java Imports

Promoted Properties

User Properties:

Name	Type	Value	Required
failOnError	Boolean	false	<input type="checkbox"/>