



# Extending WSO2 ESB 1.5

Paul Fremantle  
Co-Founder and  
VP/Technical Sales, WSO2  
[paul@wso2.com](mailto:paul@wso2.com)

# Introduction

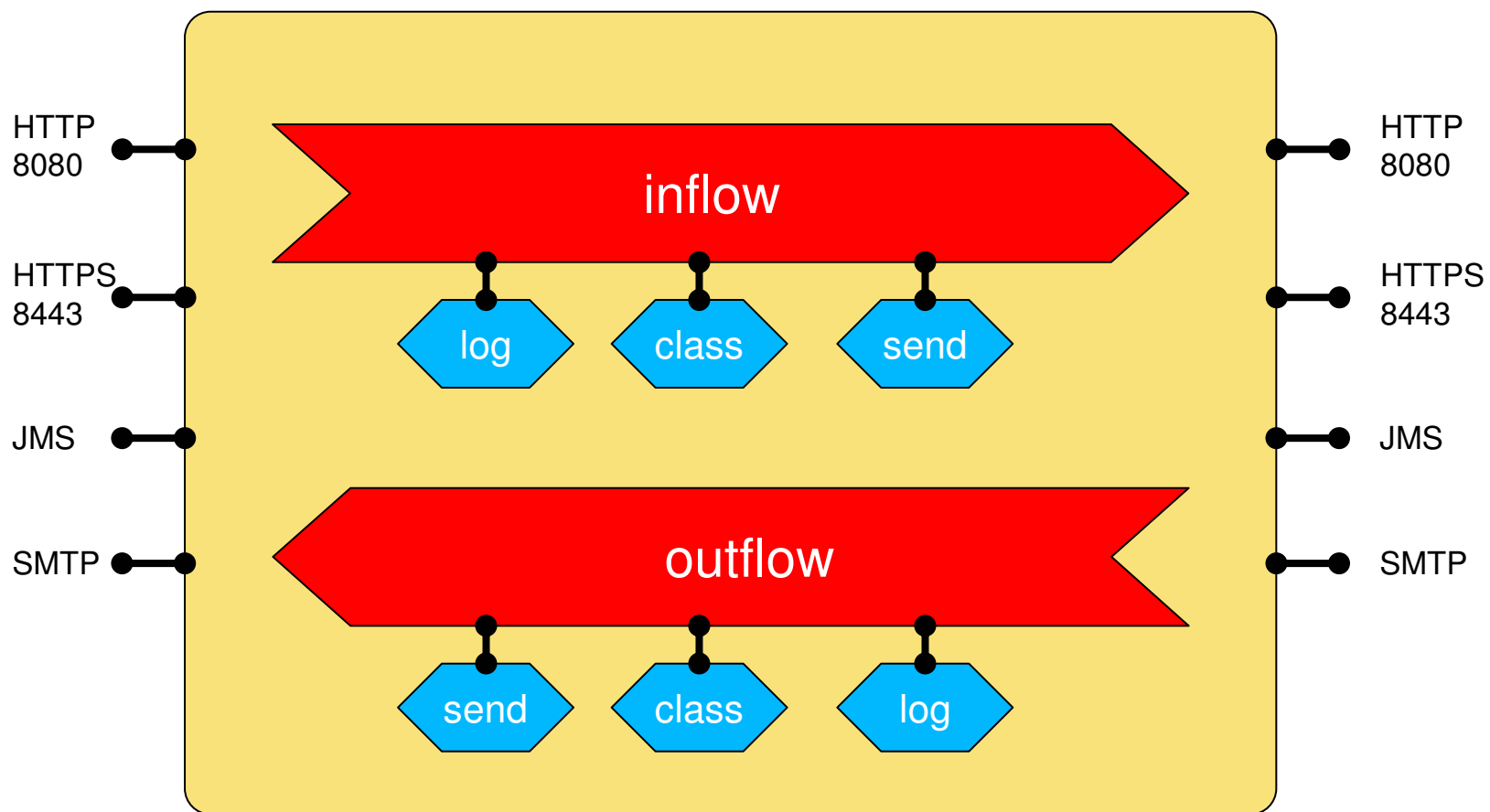
## ■ Ongoing Webinar Series

- Introducing the WSO2 ESB
  - <http://wso2.org/library/2849>
- **Understanding how to extend the WSO2 ESB**
- Adding Data Services into your SOA with WSO2 WSAS
  - 22nd January
- Event Stream Processing with WSO2 ESB and Esper
  - 19th February

# WSO2 ESB and Apache Synapse

- The WSO2 ESB is built on top of the Apache Synapse engine:
  - WSO2 ESB 1.5 is based on Synapse 1.1
  - The registry and Web UI are WSO2 ESB
  - The runtime ESB engine is 100% Synapse
- The result is that almost all of the extensions to the ESB are based on the Apache Synapse programming model
- The WebAdmin UI is 100% synced with the synapse.xml configuration
  - Edits made in the UI are propagated to the XML
  - Edits made to the XML are propagated to the UI

# Flows



# Sequence Definition

```
<inSequence>  
  <log level="full" />  
  <class name="org.fremantle.Mediator">  
    <property name="StrProp" value="Paul" />  
  </class>  
  <send />  
</inSequence>
```

# How do you write a Mediator?

- There are multiple options, with increasing levels of power
  - POJO/Commands\*
  - Script Mediators
  - Class Mediators
  - Synapse Extensions
  - UI Extensions

\*preliminary support in 1.5, full support in nightly builds and upcoming 1.6

# How do I decide which approach?

- I want to do a very simple manipulation of the message or message properties
  - *Write a POJO command*
- I am familiar with a dynamic language such as JavaScript, Ruby, Groovy and I want to use that to manipulate the message
  - *Use a script mediator*
- I am a Java programmer and I want more control over the lifecycle and full access to the message in Java
  - *Write a class mediator*
- I have written re-usable class mediators and I want to turn them into full Synapse/WSO2 ESB extensions
  - *Use the MediatorFactory model*
- I want to give administrators a UI component to configure my extensions
  - *Learn about the WSO2 ESB UI extension model*

# Writing a script mediator

- For this session I will concentrate on JavaScript / E4X
  - E4X is an ECMA standard extension to JavaScript that supports XML natively
  - No need to code DOM, SAX, StAX, etc
- But you can see my presentation on using Groovy on The ServerSide
  - see <http://pzf.fremantle.org/2007/12/making-soa-groovy-and-coding-for.html>

# E4X is very powerful

```

var payload = mc.getPayloadXML();
var symbol = payload..*::symbol[0].toString();
var search = payload..entry.(@name=="value")
mc.setPayloadXML(
    <m:getQuote xmlns:m="http://stockquote">
        <m:request>
            <m:symbol>{symbol}</m:symbol>
        </m:request>
    </m:getQuote>
);

```

XPath-like access to data inside the message

Query on values and attributes

At any point within inlined XML you can jump back to JavaScript evaluation using {}

Direct inlining of XML inside the code

# Benefits of Scripting approach

- The script can be embedded in the configuration or stored in a registry
  - No need to deploy .class files or JARs
- Very intuitive programming language for XML manipulation
  - Simpler than XSLT for many programmers
- Reasonable performance is traded off against agile development
  - Any script mediator can be rewritten as a class mediator later if performance is affected

# Class Mediators

```
<class name="org.fremantle.myMediator">
  <property name="Blah" value="hello"/>
</class>
```

- Instantiate a class
  - Just one instance across multiple messages
- Use injection to set basic or XML properties
- Then for each message calls
 

```
boolean myMediator.mediate(MessageContext mc);
```
- Gives access to the message, any properties, plus also access the overall Synapse configuration
  - return false if you want the message dropped
- Mediators may implement *ManagedLifecycle* interface
  - init / destroy allows resources to be set up and cleaned up

# PayloadHelper class

- Simplifies access to the message body

```
org.apache.synapse.util.PayloadHelper {
public static int getPayloadType(MessageContext mc)
public static OMElement getXMLPayload(MessageContext
    mc)
public static void setXMLPayload(MessageContext mc,
    OMElement element)
public static DataHandler
    getBinaryPayload(MessageContext mc)
public static void setBinaryPayload(MessageContext
    mc, DataHandler dh)
}
```

Also Text, Map, StAX (XMLStreamReader)

# Simple example: CSV->XML

```

public boolean mediate(MessageContext mc) {
    DataHandler dh = PayloadHelper.getBinaryPayload(mc);
    BufferedReader br;
    new BufferedReader(new
        InputStreamReader(dh.getInputStream()));
    CSVReader csvReader = new CSVReader(br);

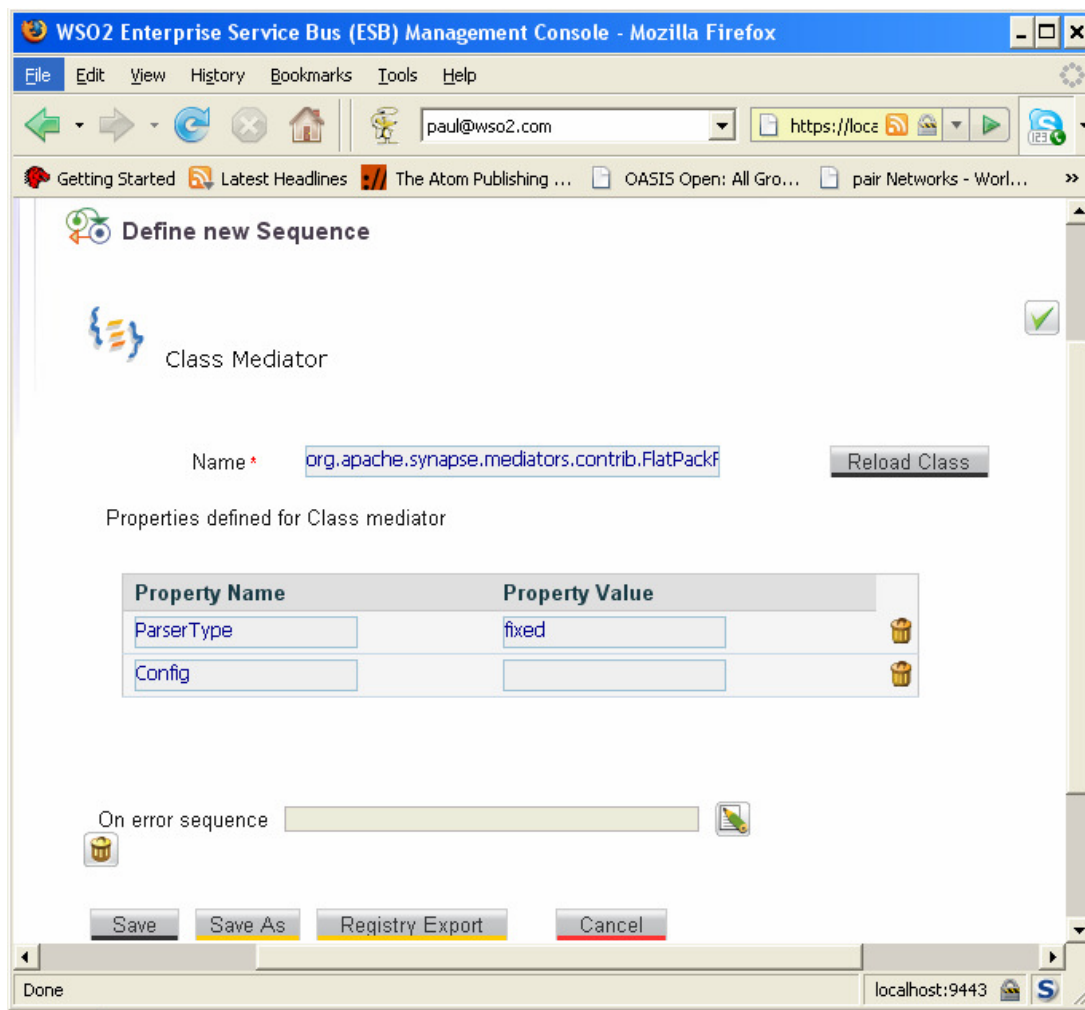
    OMFactory fac = OMAbstractFactory.getOMFactory();
    OMElement e1 = fac.createOMEElement("csv", csvNS);
    // create element to hold data
    while ((nextLine = csvReader.readNext()) != null) {
        rownum++;
        // add elements to XML
    }
    br.close();
    PayloadHelper.setXMLPayload(mc, e1);
    return true;
}

```

# The class mediator inside WSO2 ESB

- Copy the classes into the ESB classpath
  - c:\wso2esb-1.5\webapp\WEB-INF\lib
- Now create a new Sequence and choose the class mediator
- When you type in the class name and click Load, the ESB introspects your class and suggests edit fields for the settable properties

# Example



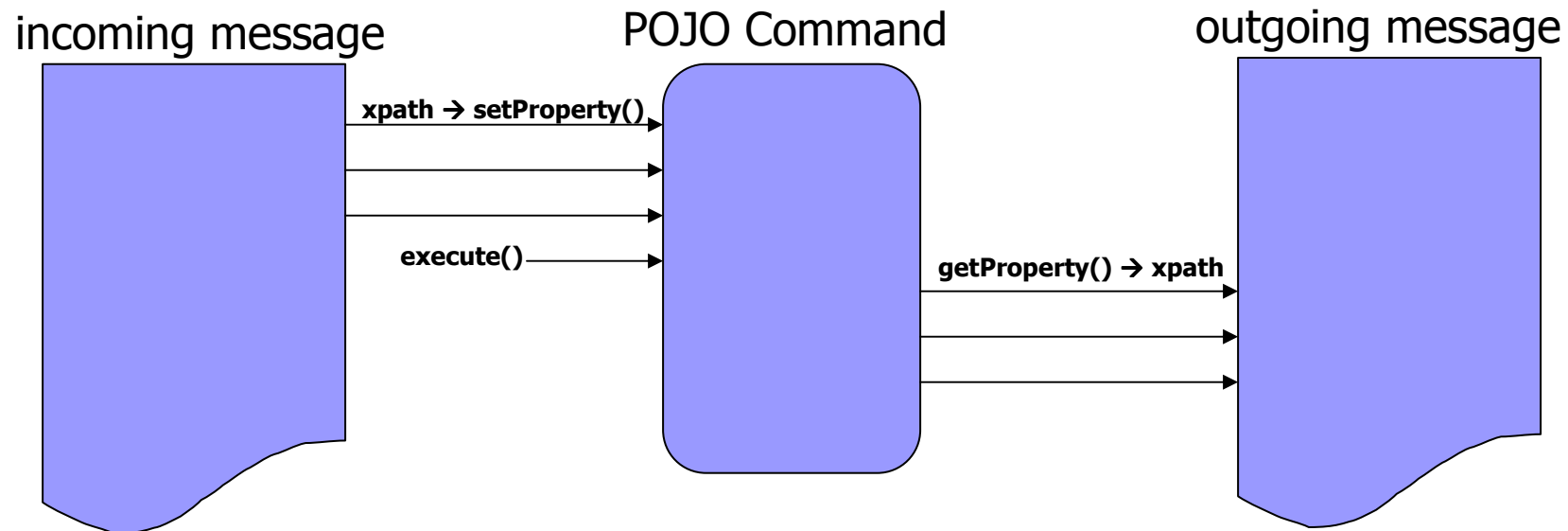
```

public class
    FlatPackFileToXML
    implements Mediator {
public void
    setParserType(String p)
    {...}
public void
    setConfig(OMEElement
    el) {...}
...
public boolean mediate()
}

```

# The POJO command model

- Based on a combination of
  - XPath
  - The Command Pattern
    - [http://en.wikipedia.org/wiki/Command\\_Pattern](http://en.wikipedia.org/wiki/Command_Pattern)



# Example of a POJO Command

```
public class PaulsCommand {  
    private String symbol, exchange;  
    public void setSymbol(String symbol) {  
        this.symbol = symbol;  
    }  
    public String getExchange() {  
        return exchange;  
    }  
    public void execute() {  
        // look up exchange in database  
        exchange = symbolToExchange(symbol);  
    }  
}
```

# POJOCommand Configuration

```

<pojoCommand name="PaulsCommand">
  <property name="symbol"
    expression="//getQuote/request/symbol"
    action="ReadFromMessage"/>
  <!-- will call setSymbol() on command -->
  <property name="quote" context-property="exchange"
    action="UpdateMessage"/>
  <!-- will update context property exchange with getExchange()
  result -->
</pojoCommand>
<switch source="get-property('exchange')">
  <case regex="NYSE">
    <!-- route to NYSE exchange -->
  </case>
  ...
</switch>

```

# Tasks

- Simple repetitive actions
- Can also be used to start a long-running activity at startup
- Uses the Quartz Scheduler to run items
  - [www.opensymphony.com/quartz](http://www.opensymphony.com/quartz)
- Tasks must implement the *Task* interface
 

```
package org.apache.synapse.startup;
public interface Task
{
    public abstract void execute();
}
```
- Tasks may implement the *ManagedLifecycle* interface
- Properties are set by injection (String and XML)

# Sample task - MessageInjector

```

public class MessageInjector implements Task,
    ManagedLifecycle
{
    public void setTo(String url)
    { to = url; }
    public void setMessage(OMElement elem)
    { message = elem; }
    public void execute() {
        MessageContext mc =
            synapseEnvironment.createMessageContext();
        mc.setTo(new EndpointReference(to));
        PayloadHelper.setXMLPayload(mc,
            message.cloneOMElement());
        synapseEnvironment.injectMessage(mc);
    }
}

```

# SynapseEnvironment

- Allows you access to create messages, inject messages, get access to a thread pool

```
public interface SynapseEnvironment {  
    public MessageContext  
        createMessageContext();  
    public boolean  
        injectMessage(MessageContext mc);  
}
```

# Task configuration

```

<task
  class="org.apache.synapse.startup.tasks.MessageInjector"
  name="inject">
  <trigger interval="5000"/>
  <property name="to"
    value="http://localhost:9000/soap/StockQuoteService"/>
  <property name="soapAction" value="urn:getQuote"/>
  <property name="message">
    <m0:getQuote xmlns:m0="http://services.samples/xsd">
      <m0:request>
        <m0:symbol>MSFT</m0:symbol>
      </m0:request>
    </m0:getQuote>
  </property>
</task>

```

# Further extensions

- Any class mediator can be extended to support its own XML configuration fragments
  - Known as *Domain Specific Modelling*
  - In fact we use this ourselves to separate out extensions
  - Can be packaged as a single JAR
    - If the JAR is in the classpath then the XML syntax can be read and written by Synapse
  - For example look at the BSF, Throttle, XQuery mediators
- The WSO2 ESB also supports extensible UI components
  - Contact us if you want help adding a UI page for your mediator

# It's not just mediators and tasks

- You can also create
  - Transports
    - Can manage incoming and outgoing protocols
  - Registry providers
    - Bridge between WSO2 ESB and a registry/repository
      - e.g. UDDI
  - Endpoints
    - For example a smart load-balancer or failover
  
- Not enough time today to go into detail
  - Contact me or
  - Send a note to [esb-java-user@wso2.org](mailto:esb-java-user@wso2.org)

# Resources

- Main project site
  - <http://wso2.org/projects/esb/java>
- Documentation
  - [http://wso2.org/project/esb/java/1.5/docs/ESB\\_Extending.html](http://wso2.org/project/esb/java/1.5/docs/ESB_Extending.html)
- Articles:
  - Writing a mediator for the WSO2 ESB
    - <http://wso2.org/library/2898>
  - Writing a Task in WSO2 ESB
    - <http://wso2.org/library/2900>
- Axiom Tutorial
  - <http://ws.apache.org/commons/axiom/OMTutorial.html>
- QuickStart guide
  - [http://wso2.org/project/esb/java/1.0/docs/ESB\\_QuickStart.html](http://wso2.org/project/esb/java/1.0/docs/ESB_QuickStart.html)
- Apache Synapse
  - <http://ws.apache.org/synapse>