Putting it All Together

Overview

This document will describe how to create a client service based on some very simple input and output needs. It will expose the user to the following:

- HTTP Post Listener
- HTTP Post Transport
- XPath Attribute Processor
- Enhanced Attribute Processor
- Groovy Processor
- Writing XSLTs

With the output generated it should provide a very realistic breadth of what the reader will need to be able to do in a normally complex implementation.

Hints and guides are provided along the way, though not nearly as much as has been provided in previous labs.

We will be using weather data in the implementation and services provided by OpenWeatherMap at http://openweathermap.org/api

To call this service, use the following app id.

AppID: a287297f3432659fc012915d0cd7d9db

Sample:

http://api.openweathermap.org/data/2.5/weather?q=<*city*>&appid=a287297f3432659fc012915d0cd7d9db

http://api.openweathermap.org/data/2.5/weather?zip=<*zip*>&appid=a287297f3432659fc012915d0cd7d9db

http://api.openweathermap.org/data/2.5/weather?id=<city_id>&appid=a287297f3432659fc012915d0cd7d9db

The incoming request (template) will be in the following format. Both City and ZipCode are optional so you may pass in city, zipcode, or both. You must pass in the AppID to use in the call to the weather service.

```
<InputData>
    <Addr>
       <City>Boston</City>
       <ZipCode>02110</ZipCode>
    </Addr>
    <AppID>a287297f3432659fc012915d0cd7d9db</AppID>
</InputData>
```

If city is provided, look up the city in a yaml file to get the city code:

- Boston: 4317656
- Dallas: 4684888
- Mumbai: 1275339
- Singapore: 1880252
- Toronto: 6167865
- Agra: 7777777

If the city code is found, call the service by city code. If the city code is not found, call the service by city name.

If no city is provided, call the service by zip code.

The weather service response is in the following json format.

```
{
"coord":{"lon":71.06,"lat":42.36},
"weather":[
     {"id":741,"main":"Fog","description":"fog","icon":"50n"},
     {"id":701,"main":"Mist","description":"mist","icon":"50n"}],
"base":"stations",
"main":{"temp":278.45,"pressure":1014,"humidity":93,"temp_min":274.15,"temp_max":286.15},
"visibility":16093,
"wind":{"speed":7.2,"deg":30,"gust":10.8},
"clouds":{"all":90},
"dt":1492931700,
"sys":{"type":1,"id":1801,"message":0.0026,"country":"US","sunrise":1492941003,"sunset":14929905
32},
"id":4930956,"name":"Boston","cod":200
}
```

Error response

```
{"cod":"404","message":"city not found"}
```

Return the weather to the originating system in the following format:

```
<c5response>
  <location>
    <city>Boston</city>
    <country>US</country>
  </location>
  <weather>
    <temp>278.45</temp>
    <pressure>1014</pressure>
    <humidity>93</humidity>
    <temp_min>274.15</temp_min>
    <temp_max>286.15</temp_max>
    <visibility>16093</visibility>
    <windspeed>7.2</windspeed>
    <sunrise>1492941003</sunrise>
    <sunset>1492990532</sunset>
  </weather>
</c5response>
```

Lab Overview

Using the Poster plugin in Firefox we'll construct an interface, which can handle fulfilling the needs of the caller.

The design should be worked through by the class and then the execution should be done by the students in parts. The reason for the progressive nature of the lab is to ensure that everyone keeps up and also to show that incremental progress is extremely important.

Watch Tower

- Have Watch Tower display the ZipCode, City, and appID values in the main table
- Have Watch Tower allow searching on the same values
    - ZipKey: Checkboxes
    - ZipCode: String
    - WeatherMode: Checkboxes
    - UserID: String

Design Thoughts

- How many interfaces will there be? Routes? Error Routes?
    - Given our contrived example, how many routes should have an Error Monitor?
- What does each route look like? How many Listeners does each have? Transports?
- What kind of attributes need to be extracted in order to make any decisions along the way? In order to create the XML appropriate for each stage?
- When conditional logic is required, what's the best way to handle it?
- What kind of formats need to be supported?
- When XSLT is needed should it be push or pull?
    - Push – good when very little changes between the "before" and "after" XML
    - Pull – good when the "after" only needs a small portion of what is available in the "before"

Setup

- Point the eiConsole to the same working directory as Eclipse is using. This will make it easier to deploy and run later. Most likely: T:\workspace\platform\web\WEB-INF\interface-root
    - Allow the eiConsole to initialize it as a Working Directory
- The client should be exposed as "http://localhost:8443/eip/http-post/weather" from Connect5
- XML Samples for the output from the client and XML to be returned have been provided in the materials:
    - serviceOutput.xml – Used for Source sample
        - returnFormat.xml – Used for Target sample for return data
- Run the following script to create a new table to be used by the DB Monitor step

```
CREATE TABLE TR_WEATHER_LOG (
    ID MEDIUMINT PRIMARY KEY AUTO_INCREMENT,
    ZIP_CODE CHAR(5),
    CITY VARCHAR(128),
    APP_ID VARCHAR(32),
    TIME_STAMP TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Tasks

- The Message ID that gets used is sequential by default. Each time the application restarts so does that count. This can cause issues as logs will get grouped together when they shouldn't. Use a Processor to solve this. See the "References" Section for additional notes.
    - Normally we would rely on the MessageID in the input to be universally unique, but Firefox's Poster cannot generate one of those on the fly.

References

- *Attribute which is used for Message ID:* com.pilotfish.eip.OriginatingTransactionID
- *Java for generating a UUID:* @java.util.UUID@randomUUID()
- XSLT Identity Transform (with PilotFish Converter initialized; lightweight version)

```
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns=""
    xmlns:converter="xalan://com.pilotfish.eip.transform.ConverterProxy"
    exclude-result-prefixes="converter"
    extension-element-prefixes="converter"
    version="1.0">

  <xsl:output method="xml" indent="yes" />

  <!--
    Since we know the root will only match once, it's the ideal place
      to initialize the converter.
    -->
  <xsl:template match="/">
    <converter:register class="com.pilotfish.eip.transform.converters.EchoConverter"
name="EchoConveter" parms="EchoField">
      <xsl:element name="Mode">
        <xsl:text>Mode</xsl:text>
      </xsl:element>
    </converter:register>
   <xsl:apply-templates select="@*|node()" />
  </xsl:template>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```