ORACLE
INSURANCE

Oracle® Insurance Policy Administration

Web Services

Version 9.2.0.0.0

Part number: E16287-01

March 2010

Table of Contents

WEB SERVICES OVERVIEW

The Oracle Insurance Policy Administration (OIPA) system exposes many of its system's functionalities to external applications through web services. The available exposed services are as follows:

- FileReceived – Service for inserting data and providing quote details.
- ExposedComputation – Service for exposing OIPA's math engine.
- PASProperties – Service for retrieving system configuration properties.

This document not only discusses the available web services, but also gives basic overviews of protocols and demonstrates creating messages needed by the web services for a more holistic explanation of the available functionality. Note that this documentation uses SOAP messages as a means to explain functionality.

CREATING MESSAGES WITH SOAP

SOAP OVERVIEW

SOAP (Simple Object Access Protocol) is a XML-based language used for the transport of structured information from a requester to a provider. A SOAP message is sent from the requesting application to an OIPA web service. The web service will interpret the data and write it to the database as specified in the message. A SOAP response message including the outcome is then returned to the requester.

In the context of OIPA, a SOAP message can be sent using HTTP or HTTPS, for added security. The proper authentication information must be included in the security portion of the header. The body, explained in detail later in this document, simply consists of the processFileReceivedRequest message, as defined by the service's WSDL.

WSDL

WSDL (Web Service Definition Language) is a XML-based language used to describe web services. In case of OIPA, the WSDL for each available web service defines the message format, data type, and transport protocols to be used between the requester and OIPA, the provider.

A listing of all available web services and their associated WSDL can be found here:

http://<OIPA-SERVER:9080>/PASJava/service/

Security

OIPA adheres to the WS-Security standards for the authentication of SOAP messages.

The standards, as developed by the OASIS Open committee, can be referenced here:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

The `<wsse:UsernameToken>` element is used to contain the authentication information. The username and password are specified inside of the `<wsse:Username>`, and `<wsse:Password>` elements, respectively.

It is suggested that SSL (Secure Socket Layer) is used as a method of encryption for all SOAP messages.

The optional `<wsse:Nonce>` element allows for the usage of a nonce as added security. A *nonce* is a random number, in this case represented in base 64, which is embedded in the security header to aid in preventing old communications from being reused. This number is newly generated for each request on the client side and is returned along with the SOAP response from OIPA. The `<wsu:Created>` element must contain the timestamp of the creation time of the nonce.

```
<soapenv:Header>
     <wsse:Security
       soapenv:mustUnderstand="1"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
          wssecurity-secext-1.0.xsd">
       <wsse:UsernameToken
            wsu:Id="UsernameToken-1"
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
             200401-wss-wssecurity-utility-1.0.xsd">
```

```
            <wsse:Username>username</wsse:Username>
            <wsse:Password
                Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
                   wss-username-token-profile-1.0#PasswordText">password
                   </wsse:Password>
            <wsse:Nonce
                EncodingType="http://docs.oasis-
                   open.org/wss/2004/01/oasis-200401-wss-soap-message-
                   security-1.0#Base64Binary">
                   OUtRdmO7dLg/v+0DI04/DA==</wsse:Nonce>
            <wsu:Created>
                2009-09-28T17:43:02.546Z</wsu:Created>
        </wsse:UsernameToken>
    </wsse:Security>
</soapenv:Header>
```

OIPA WEB SERVICES

PAS PROPERTIES

The PAS Properties web service is exposed for an external application to retrieve all property values from the PAS.properties configuration file.

It is available using the "getPASProperties" operation on the wsdl "…/PASJava/service/PASProperties?wsdl".

There are no required parameters. The response XML from a call to this service has "PasProperties" as the root element. Then there is a child "Property" element for each property in the file. Below is a sample response from calling this service (not all properties are included):

```
<PasProperties>
     <Property NAME="application.defaultCurrencyCode">USD</Property>
     <Property NAME="application.databaseType">Oracle</Property>
     <Property NAME="transaction.manager">jpa</Property>
</PasProperties>
```

FILERECEIVED

Introduction

The OIPA FileReceived web service allows for an application to send records to be written to tables in the OIPA database. A SOAP message is sent by the application to the FileReceived web service. Note: This is also called by the name AsFile by project team members.

The SOAP message includes two OIPA-specific elements, the FileID and the XML element. The FileID element identifies the configuration for OIPA to use when transforming the inbound XML into OIPA's AsXml. The XML element includes the data to be written to the OIPA database.

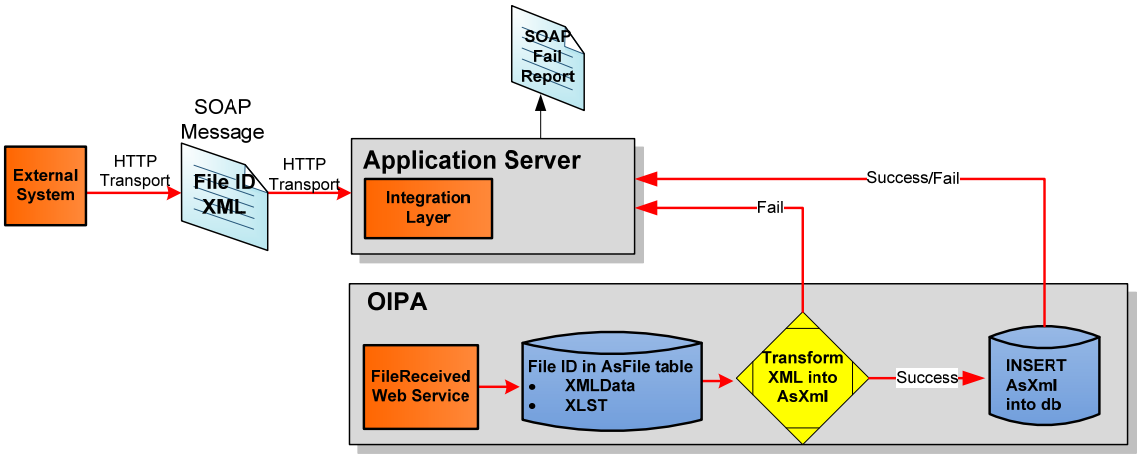
Once the SOAP message is received by the FileReceived web service, attributes needed to complete the request are assigned values as configured in XMLData. The data in the XML element is then transformed into AsXml using the configured XSLT. The result of this transformation is then written to the database.

A SOAP message is sent back to the requestor, or caller, including the result of the request. If the request was successful, the message will consist of the transformed AsXml. If the request was not successful, the message will consist of a SOAP Fault detailing any possible errors.

The WSDL for the FileReceived service is located here:

http:/</OIPA-SERVER:9080>/PASJava/service/FileReceived?wsdl

High Level Flow Overview



Inbound Flow

1) FileReceived Web Service receives a request via a SOAP message
2) The File is looked up using the FileID specified by the request.
3) The XMLData's AssignAttributes XML is processed.
4) The XSLT maps the request XML to AsXml.
5) The transformed AsXml is mapped to data objects.
6) PreInsert operations are performed on the objects.
7) The objects are inserted into the database.
8) PostInsert operations are performed on the objects.
9) The transformed AsXml is returned to the caller.

An example of this inbound message would be the addition of an activity to a policy in the OIPA system. Such a need might arise in the case of integration with a new business system. When approved a message might be sent to the OIPA system to create a new copy of a policy.

AsFile Overview

The AsFile database table stores the user-configurable portion of the FileReceived web service. The table contains a separate record for each type of file OIPA has been configured to receive.

The table includes six columns, FileGUID, CompanyGUID, FileNameFormat, FileID, XSLT, and XMLData. The FileNameFormat column stores the descriptive name of the file format type.

The FileID column stores a unique three-character ID used to describe the file format. An inbound SOAP message will include a <FileID> element specifying the format of the file being sent.

The XMLData column stores the XML configuration for each specific File business rule.

The XSLT column stores the XSLT used to transform the inbound XML into AsXml.

XMLData

The File business rule allows for values to be assigned to various attributes before the inbound XML undergoes the transformation process into AsXml. This is configured by using the <Attribute> element inside of the <AssignAttribute> parent element to assign values. By assigning values at this stage of the process, preexisting data from OIPA can be used to populate the AsXml.

<AssignAttributes>

The <AssignAttributes> element is the parent element which contains <Attribute> elements.

<Attribute>

Any processing of data that needs to take place prior to the transformation process is done using the <Attribute> element. The element has two attributes, NAME and TYPE. NAME specifies the name of the attribute, while TYPE defines how the expression will be evaluated. Attributes are evaluated from the top down, so attributes listed first can be used in expressions below them.

For example, the GUID type will set the attribute to a newly generated GUID, and the XPATH attribute will parse data from the inbound XML so that it may be manipulated prior to transformation.

All Attributes that will be used in the XSLT stylesheet must also be defined in the XSLT stylesheet. This is explained in the XSLT section.

The following table lists the available TYPES:

TYPE	Description
GUID	Sets the attribute to a newly generated GUID.
VALUE	Sets the attribute to the specified value.
SYSTEMDATE	Sets the attribute to the current system date.
SEQUENCE	Sets the attribute by calling asc_NextSequenceInteger and passing the NAME as a parameter.
XPATH	Sets the attribute to the result of the specified XPATH expression
XPATHSTRINGLIST	Sets the attribute to a comma delimited list containing the resulting values of the XPATH
XPATHNUMBERLIST	Sets the attribute to a comma delimited list containing the resulting values of the XPATH
SQL	Sets the attribute to the result of the specified SQL statement
SQLMAP	Sets the attribute to a 'key-value-pair' type collection of the resulting values of the SQL Statement

Examples:

GUID:

```
<Attribute NAME="PolicyGUID" TYPE="GUID"></Attribute>
```

XPATH:

```
<Attribute NAME="Field" TYPE="XPATH">/Request/PolicyName</Attribute>
```

<PreInsert> and <PostInsert>

<PreInsert> and <Post Insert> are optional elements that allow for other system functionally to be called before or after the data is inserted into the database. This is done by calling specific types of Java classes which are used for these operations. The architecture of the Pre and Post Insert functionality allows for these classes to be dynamically instantiated at runtime

Pre and Post Insert operations are specified in the XMLData portion of a File's configuration, after the closing of the AssignAttributes element. The CLASS attribute of both elements allows for setting the name of the Java class to be called.

The following example will invoke the AsFile Post Insert Activity Processor

after the records are inserted into the database:

```
<PostInsert>
```

```
    <Object CLASS="com.adminserver.pas.webservice.bll.AsFilePost
   InsertIndividualActivityProcessorBll">
    </Object>
</PostInsert>
```

XSLT

Overview

XSLT (Extensible Stylesheet Language) is a XML-based language used for the transformation of XML documents into other XML documents. Using XSLT, OIPA transforms the inbound payload of the SOAP message into AsXml, which can then be processed by the system.

OIPA adheres to XSLT Version 2 specifications which allows for very flexible configuration of the transformation process. Standard XSLT elements can be used to transform the inbound message into AsXml based on templates, perform data validation and error handling.

Using Attributes from XMLData

Each Attribute defined in the XMLData section that will be needed in the XSLT stylesheet must first be declared as a parameter in the XSLT.

As an example, if an Attribute named PolicyGuid was set in the XMLData section, and it's value needs to be referenced in the XSLT, the following would need to be added in the beginning of the XSLT, directly proceeding the prolog.

```
<xsl:param name="PolicyGuid"/>
```

Functions

Several functions are available for use inside of the XSLT stylesheet which allow for added functionally such as generating GUIDs and retrieving the current system time. The available utility functions are:

- getNextGUID()
- getGmtTime()
- formatDateTime()
- addMillis()

In order to use these added functions, the XsltFunctionHelper class must be added as a namespace in the XSLT prolog as noted below.

```
    xmlns:utl="com.adminserver.webservice.helper.XsltFunctionHelper"
```

The getNextGUID() function will generate a new GUID. As an example, the following code would output a newly generated GUID inside of the <PolicyGuid> element.

```
<xsl:element name="PolicyGuid">
    <xsl:value-of select="utl:getNextGUID()"/>
</xsl:element>
```

Functions can also be used to retrieve the current system time and then format it properly for insertion into the database.

```
<xsl:template name="GMT">
    <xsl:param name="Offset" select="0" as="xs:integer"/>
    <xsl:value-of select="utl:formatDateTime(utl:addMillis
  (utl:getGmtTime(), $Offset))"/>
</xsl:template>
```

Validation and Error Handling

AsFile has the ability to perform data validations using XSLT portion of the configuration. As an example, the value of a variable can be tested to ensure the value is as expected.

The following validation syntax can be used anywhere in the XSLT:

```
<xsl:if test="$variable = 'incorrect value'">
     <xsl:variable name="Error1" select="Error Message"/>
     <ValidationError ERRORSTATUSCODE="Err001">
       <xsl:value-of select="$Error1 "/>
     </ValidationError>
</xsl:if>
```

If upon evaluation the `<xsl:if>` expression is true, the `<ValidationError>` element will be included in the resulting AsXml.

If the one or more `<ValidationError>` elements exist in the AsXml, a SOAP fault will be thrown and the text within the element and the ERRORSTATUSCODE will be returned to the caller as part of the SOAP fault.

A Transformation Example

XML Portion of SOAP Request

```
<Request>
     <PolicyName>TestPolicy</PolicyName>
</Request>
```

XSLT

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="2.0">

     <xsl:template match="/Request">
       <AsXml>
            <AsPolicy>
                <PolicyName>
                   <xsl:value-of select="/Request/PolicyName"/>
                </PolicyName>
            </AsPolicy>
       </AsXml>
     </xsl:template>
     </xsl:stylesheet>
```

Resulting AsXml after transformation:

```
<AsXml>
     <AsPolicy>
       <PolicyName>
            TestPolicy
       </PolicyName>
     </AsPolicy>
</AsXml>
```

AsXml

AsXml is the XML formatting used by OIPA to store data destined for the database. The format is very simplistic, including a parent element for each database table, and a child element for each column of the table. The root element, `<AsXml>`, must be used to identify the formatting.

Each element must exactly match the name of the table or column in the database for mapping purposes. Each column of the table needs to be populated with data unless the column is set to allow NULL.

Schema

```
<AsXml>
     <Table>
       <Column>Value</Column>
     </Table>
</AsXml>
```

Examples

Example of a record from AsPolicy in AsXml format:

```
<AsXml>
     <AsPolicy>
       <PolicyGuid>6CCA0B15-EFAC-471F-A698-27949AB9B9C4</PolicyGuid>
       <PlanGuid>3904A440-E035-40A1-9905-D544F7A6C093</PlanGuid>
       <CompanyGuid>A9211F9D-2C3B-4523-8151-768684696488</CompanyGuid>
       <PolicyNumber>GLPT31012265</PolicyNumber>
       PolicyName>Term Policy</PolicyName>
       <IssueStateCode>38</IssueStateCode>
       <PlanDate>1/29/2031 12:00:00 AM</PlanDate>
       <UpdatedGmt>9/10/2009 6:43:01 PM</UpdatedGmt>
       <StatusCode>09</StatusCode>
       <CreationDate>1/29/2031 12:00:00 AM</CreationDate>
       <XmlData></XmlData>
     </AsPolicy>
</AsXml>
```

Example of a record from AsRate in AsXml format:

```
<AsXml>
     <AsRate>
       <RateGuid>CA132F95-E768-45AE-ABBE-00000980034B</RateGuid>
       <RateGroupGuid> C6496E59-7EF7-4719-959B2C0065CA4EF9</RateGroupGuid>
       <RateDescription>Term_20_Premium<RateDescription>
       <Criteria1>03</Criteria1>
       <Criteria2>02</Criteria2>
       <IntegerCriteria>12</IntegerCriteria>
       <Rate>11.35</Rate>
     <AsRate>
</AsXml>
```

EXPOSED COMPUTATION

Overview

The Exposed Computation web service is exposed to give an external application access to OIPA's robust math engine. An exposed computation can execute a calculation and return result values in the SOAP response XML.

The web service is available by using the "processExposedComputation" operation of the ExposedComputation web service. The WSDL is located at: "/PASJava/service/ExposedComputation?wsdl".

A request to this service takes two text parameters:

- **ComputationID** – Identifier for a record in the AsExposedComputation table.
- **XML** – A text parameter which is treated as an xml document by the service.

The service processes a math engine based on the configuration for the passed ComputationID parameter. The XML document can be used for feeding extra dynamic parameters to a request. The response from the call is also driven by the configuration of the ComputationID in the request.

Request Flow

The ExposedComputation web service processes a request with the following steps:

1) Load the AsExposedComputation record for the ComputationID from the request.
2) Parse the parameters from the incoming xml document, and set policy context if policy information is supplied.
3) Load exposed computation business rule (from AsBusinessRules).
4) Do valuation if the exposed computation business rule is configured to do so.
5) Execute math from the Input configuration in the exposed computation business rule.
6) Build response xml from the Output configuration in the exposed computation business rule.

AsExposedComputation Table

The AsExposedComputation table is a configuration table for processing exposed computations. An AsExposedComputation record will have a unique ComputationID value. The other relevent coluimn in this table is the RuleName column which is the name of the rule in the AsBusinessRules table that contains the configuration for processing.

Exposed Computation Business Rule

The RuleName from the AsExposedComputation for the exposed computation request is used to load an AsBusinessRule record that contains the configuration for how to process. The Exposed Computation business rule is setup in a similar way to how the Calculate business rules are configured for calculating segments. There is an Input element which contains the math variables configuration for processing the math engine. The math variables are to be configured the same way as any other math section in the system. Then, there is an Output element which contains the mappings for the input variables to output in the response. Below is a sample of a simple exposed computation configuration:

```
<ExposedComputation>
     <Input>
       <MathVariables>
            <MathVariable VARIABLENAME="Variable1"
            TYPE="VALUE" DATATYPE="TEXT">
            TestValue</MathVariable>
```

```
      </MathVariables>
    </Input>
    <Output>
      <Mappings>
            <Mapping OUTPUTNAME="Result1">Variable1</Mapping>
      </Mappings>
    </Output>
</ExposedComputation>
```

The Input element contains the math to process, and the Output element pulls the one math variable into the output for “Result1”. The response of the exposed computation request will contain the value for “Result1”.

Valuation

The exposed computation also has the ability to do valuation. The configuration for this support is below:

```
<ExposedComputation VALUATION=”Yes”>
    …
</ExposedComputation>
```

By adding this attribute, valuation is run before executing math. This makes available all valuation FIELDs (Valuation:Policy:CashValue, Valuation:Fund:FundGUID:CashValue, etc.) to the math configuration. Valuation can only be executed if the request is being processed in the context of a policy.

When processing valuation that may contain variable funds, there is also the ability for using the nearest nuvs for the funds. This is achieved through the following configuration:

```
<ExposedComputation VALUATION=”Yes” NEARESTNUV=”Yes”>
    …
</ExposedComputation>
```

Exposed Computation SOAP Messages

SOAP Request Input Parameters

When making an ExposedComputation request, the second parameter available is a String that the service treats as an XML document. This XML contains parameters that can be used as FIELD values when executing the math engine.

PolicyNumber

Inclusion of a “PolicyNumber” parameter tells the exposed computation that it is being processed for a policy. The exposed computation business rule is now overridable by plan with this parameter, and the math engine will have access to Policy and Plan FIELD variables for that policy when executed. This parameter is also required if the exposed computation is configured to do valuation during the request.

EffectiveDate

Inclusion of an “EffectiveDate” parameter can be done when an exposed computation executes valuation during the request. This date will be used as the valuation date during valuation. If this parameter is not defined and valuation is still executed, the valuation date will default to the system date.

Below is the expected format for the XML parameter in the request:

```
<Parameters>
    <Parameter NAME=”PolicyNumber”>POL12345</Parameter>
    <Parameter NAME=”EffectiveDate”>01/01/2009</Parameter>
</Parameters>
```

SOAP Response

The data returned in the SOAP response xml from a call to ExposedComputation is built from the <Output> mappings configured in the exposed computation rule. The root element from the response is the ComputationID from the request. Each child element of the root is the Mapping from the output configuration with its math value as the element text. For the ComputationID “EC_Test” with the below exposed computation configuration:

```
<ExposedComputation>
     <Input>
       <MathVariables>
            <MathVariable VARIABLENAME="Variable1"
              TYPE="VALUE" DATATYPE="TEXT">
              TestValue1</MathVariable>
            <MathVariable VARIABLENAME="Variable2"
              TYPE="VALUE" ATATYPE="TEXT">
              TestValue2</MathVariable>
       </MathVariables>
     </Input>
     <Output>
       <Mappings>
            <Mapping OUTPUTNAME="Result1">Variable1</Mapping>
            <Mapping OUTPUTNAME="Result2">Variable2</Mapping>
       </Mappings>
     </Output>
</ExposedComputation>
```

This will be the response xml:

```
<EC_Test>
     <Result1>TestValue1</Result1>
     <Result2>TestValue2</Result2>
</EC_Test>
```

ADDITIONAL FILE RECEIVED EXAMPLES USING ACCORD LOMA

Each of the following examples is meant to show how the various configuration files can be used to configure the FileReceived web service to integrate with other systems. A majority of these examples are from the Accord 103 implementation which is included in version 9.1 of OIPA.

SOAP REQUEST

Full Request

The SOAP request message must include an element indicating the service to use, processFileReceived in the case of the FileReceived web service. Inside of this parent element two child elements need to be included, the first declaring the corresponding FileID in AsFile, and the second including the XML destined for transformation.

The `<![CDATA[ ]]>` section allows for us to pass the full XML message to the web service without it being evaluated by the initial parser of the SOAP message. If the usage of CDATA is not desired, all characters which may be misinterpreted, such as "<" and "&", must replaced with their respective escape sequences, "`<`" and "`&`" in this case.

```
<soapenv:Envelope
     xmlns:fil="http://filereceived.webservice.adminserver.com"
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <soapenv:Header>
       <wsse:Security soapenv:mustUnderstand="1"
          xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
          wss-wssecurity-secext-1.0.xsd">
             <wsse:UsernameToken wsu:Id="UsernameToken-6"
               xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
               200401-wss-wssecurity-utility-1.0.xsd">
                  <wsse:Username>install</wsse:Username>
                  <wsse:Password Type="http://docs.oasis-open.org/wss/
                     2004/01/oasis-200401-wss-username-token-profile-
                     1.0#PasswordText">install</wsse:Password>
             </wsse:UsernameToken>
       </wsse:Security>
     </soapenv:Header>
     <soapenv:Body>
       <fil:processFileReceived
            soapenv:encodingStyle="http://schemas.xmlsoap.org/
              soap/encoding/">
            <fileId xsi:type="soapenc:string">TE1</fileId>
            <xml xsi:type="soapenc:string">
                <![CDATA[
                   <Request>
                      <PolicyName>TestPolicy</PolicyName>
                   </Request>
                ]]>
            </xml>
```

```
      </fil:processFileReceived>
    </soapenv:Body>
</soapenv:Envelope>
```

XML Message

This example shows a very small portion of a sample request that follows the Acord 103 specification. You can see in this example that the information can be nested in a structured manner as needed. Each element can have attributes to aid in clarifying data.

```
<TXLife>
 <TXLifeRequest>
  <OLife>
   <Holding id="Holding_1">
    <HoldingTypeCode tc="2">Policy</HoldingTypeCode>
    <Purpose tc="21">Family Income</Purpose>
    <Policy>
     <LineOfBusiness tc="1">Life</LineOfBusiness>
     <ProductCode>F34523A4-7988-48E0-BED9-BE2CF82FFC5F</ProductCode>
     <PolicyStatus tc="21">Applied For</PolicyStatus>
     <IssueType tc="1">Full Underwriting</IssueType>
     <Jurisdiction tc="45">Pennsylvania</Jurisdiction>
     <ReplacementType tc="1">None</ReplacementType>
     <IssueDate>2008-02-15</IssueDate>
     <PaymentMode tc="1">Annual</PaymentMode>
     <PaymentMethod tc="2">Regular Billing</PaymentMethod>
     <Life>
      <QualPlanType tc="1">NonQualified</QualPlanType>
      <Coverage id="BaseCoverage">
      <PlanName>Acme Term</PlanName>
      <ProductCode>04</ProductCode>
      <LifeCovTypeCode tc="06">Term Life</LifeCovTypeCode>
      <IndicatorCode tc="1">Base</IndicatorCode>
      <LivesType tc="1">Single Life</LivesType>
      <QualAddBenefitInd tc="1">True</QualAddBenefitInd>
      <InitCovAmt>1000000</InitCovAmt>
      <EffDate>2008-02-15</EffDate>
     </Life>
    </Policy>
   </Holding>
  </OLife>
 </TXLifeRequest>
</TXLife>
```

SOAP RESPONSE

The SOAP Response is the message that OIPA returns to the caller after receiving a SOAP request. There are two possible outcomes to a SOAP request, a success or a fault.

Successful Response

Upon successful processing of an inbound SOAP request a SOAP response is returned to the caller along with the transformed AsXml which was inserted into the database. This default configuration can be changed and the

data returned to the caller can be modified by inserting a XSLT stylesheet in to the AsFileOutput table of the database. This example shows the default SOAP response.

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <AsXml>
       <AsRate>
      <RateGuid>CA132F95-E768-45AE-ABBE-00000980034B</RateGuid>
      <RateGroupGuid>
         C6496E59-7EF7-4719-959B2C0065CA4EF9</RateGroupGuid>
      <RateDescription>Term_20_Premium<RateDescription>
      <Criteria1>03</Criteria1>
      <Criteria2>02</Criteria2>
      <IntegerCriteria>12</IntegerCriteria>
      <Rate>11.35</Rate>
       <AsRate>
     </AsXml>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP Fault

If, for any reason, there is an error while processing the inbound SOAP request, OIPA will return a SOAP Fault response message along with detail surrounding the error. In this example, a SOAP Fault message is being returned because the security parameters sent in the SOAP request were incorrect.

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>ns1:Receiver</faultcode>
         <faultstring>Authorization failed.</faultstring>
         <detail>
            <ns2:AsErrorDetail>
               <ns2:Error TYPE="System">
                  <ns2:Message>Authorization failed.</ns2:Message>
               </ns2:Error>
            </ns2:AsErrorDetail>
            <ns3:hostname>WS-Training</ns3:hostname>
         </detail>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

XMLDATA

Any data that is needed prior to the XSLT transformation process can be processed in the XMLData section of the File business rule. This example shows the use the XPATH, which allows for data from the incoming

request to be manipulated. Also illustrated in this example is the GUID Attribute type, which automatically sets the Attribute value to a newly generated GUID.

```
<File>
   <AssignAttributes>
      <Attribute NAME="PlanGUID" TYPE="XPATH">
         /TXLife/TXLifeRequest/OLifE/Holding/Policy/ProductCode
      </Attribute>
      <Attribute NAME="PolicyGUID" TYPE="GUID"></Attribute>
      <Attribute NAME="CompanyGUID" TYPE="VALUE">
      18B611A8-4429-4C67-94E6-3F4A882C9A8D</Attribute>
   </AssignAttributes>
   <PostInsert>
      <Object
   CLASS="com.adminserver.utl.AsFilePostInsertActivityProcessorUtl">
      </Object>
   </PostInsert>
</File>
```

XSLT

Format

Each XSLT stylesheet used must include a prolog defining the XSLT namespace and version. The stylesheet must also create the necessary database table structure inside the <AsXml> element as shown.

```
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"    version="2.0">

   <xsl:template match="/">
       <AsXml>
       <AsPolicy>
          <PolicyName>
          <xsl:value-of select="/Request/PolicyName"/>
          </PolicyName>
       </AsPolicy>
       <AsXml>
   </xsl:template>
</xsl:stylesheet>
```

Templates

A XSLT stylesheet can have several templates that allow for setting rules to be used during the transformation process. The following example shows the construction of the AsPolicy database table.

```
<xsl:template match="TXLife">
 <xsl:element name="AsXml">
   <!-- Create the AsPolicy Record -->
   <xsl:comment>Policy Info</xsl:comment>
   <xsl:element name="AsPolicy">
      <xsl:element name="PolicyGuid">
        <xsl:value-of select="$PolicyGUID"></xsl:value-of>
      </xsl:element>
      <xsl:element name="StatusCode">
         <xsl:text>08</xsl:text>
      </xsl:element>
```

```
      <xsl:element name="IssueStateCode">
         <xsl:value-of select="$StateCode"></xsl:value-of>
      </xsl:element>
      <xsl:element name="PolicyNumber">
         <xsl:value-of select="$PolicyNumber"></xsl:value-of>
      </xsl:element>
      <xsl:element name="PlanDate">
         <xsl:value-of select="$PlanDate"></xsl:value-of>
      </xsl:element>
      <xsl:element name="PlanGuid">
         <xsl:value-of select="$PlanGUID"></xsl:value-of>
      </xsl:element>
      <xsl:element name="CompanyGuid">
         <xsl:value-of select="$CompanyGUID"></xsl:value-of>
      </xsl:element>
      <xsl:element name="UpdatedGMT">
         <xsl:value-of select="$UpdatedGMT"></xsl:value-of>
      </xsl:element>
      <xsl:element name="CreationDate">
         <xsl:value-of select="$CreationDate"></xsl:value-of>
      </xsl:element>
     </xsl:element>
   </xsl:element>
</xsl:template>
```

Data Validation

The XSLT style sheet can be used to test information in the inbound request and make any changes necessary to be compactable with OIPA. The following example tests the Code parameter for certain possibilities and then outputs the correct gender code, M or F, as understood by the OIPA chassis.

```
<xsl:template name="getGenderCode">
   <xsl:param name="Code"></xsl:param>
   <xsl:choose>
      <!--Male -->
      <xsl:when test="$Code = '1'">
         <xsl:text>M</xsl:text>
      </xsl:when>
      <xsl:when test="$Code = 'Male'">
         <xsl:text>M</xsl:text>
      </xsl:when>
      <!-- Female -->
      <xsl:when test="$Code = '2'">
         <xsl:text>F</xsl:text>
      </xsl:when>
      <xsl:when test="$Code = 'Female'">
         <xsl:text>F</xsl:text>
      </xsl:when>
   </xsl:choose>
</xsl:template>
```