ORACLE
INSURANCE

Oracle® Insurance Policy Administration

Web Services

Version 9.7.1.0

Documentation Part Number: E51561-01

December, 2013

License Restrictions

ORACLE
INSURANCE

Table of Contents

Deleted: 5
Deleted: 5
Deleted: 6
Deleted: 6
Deleted: 6
Deleted: 7
Deleted: 8
Deleted: 8
Deleted: 8
Deleted: 9
Deleted: 9
Deleted: 9
Deleted: 10
Deleted: 11
Deleted: 17
Deleted: 19
Deleted: 19
Deleted: 19
Deleted: 19
Deleted: 20
Deleted: 21
Deleted: 22
Deleted: 22
Deleted: 25
Deleted: 25
Deleted: 25
Deleted: 26
Deleted: 26
Deleted: 27
Deleted: 27
Deleted: 29
Deleted: 29
Deleted: 29
Deleted: 29
Deleted: 30
Deleted: 31
Deleted: 31
Deleted: 32

WEB SERVICES OVERVIEW

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

- ValuePolicy – Service for running valuations on a policy.
- FileReceived – Service for inserting data and providing quote details.
- ProcessPolicy – Service for processing all pending activities on an existing policy.
- ExposedComputation – Service for exposing OIPA's math engine.

This document not only discusses the available Web Services, but also gives a basic overview of protocols and demonstrates the process for creating messages needed by the Web Services for a more holistic explanation of the available functionality.

Note:	This documentation uses SOAP messages as a means to explain functionality.

Customer Support

If you have any questions about the installation or use of our products, please visit the My Oracle Support website: https://support.oracle.com, or call (800) 223-1711.

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

CREATING MESSAGES WITH SOAP

SOAP Overview

SOAP (Simple Object Access Protocol) is an 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.

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. 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 message, as defined by the service's WSDL.

WSDL

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

A list of all available Web Services and their associated WSDLs are listed below, where the server name "server.domain.tld" and the port number "9080" are specific to each deployment of OIPA.

Note:	This document assumes the OIPA WAR name is PASJava.war.

- FileReceived: http://server.domain.tld:9080/PASJava/FileReceived?wsdl
- ExposedComputation: http://server.domain.tld:9080/PASJava/ExposedComputation?wsdl
- ValuePolicy: http://server.domain.tld:9080/PASJava/ValuePolicy?wsdl
- ProcessPolicy: http://server.domain.tld:9080/PASJava/ProcessPolicy?wsdl

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 include 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) be 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#PasswordDigest">EncryptedPassword</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>
```

INBOUND WEB SERVICES

ValuePolicy

The ValuePolicy Web Service allows valuation to be run on an already existing policy. An inputXml will be passed in with values for PolicyNumber, PolicyValuesFlag, EffectiveDate, and Nearest. Below is a sample inputXml.

```
<Parameters>
    <Parameter NAME=”PolicyNumber”>PolicyNumber</Parameter>
    <Parameter NAME="PolicyValuesFlag">Yes|No</Parameter>
    <Parameter NAME="Nearest">Yes|No</Parameter>
    <Parameter NAME="EffectiveDate">[Date]</Parameter>
</Parameters>
```

SOAP Request Example

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:val="http://ValuePolicy">
   <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>tester</wsse:Username>
            <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordText">tester</wsse:Password>
            <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary">FRkBB/REsT/4ThQzEjoiUQ==</wsse:Nonce>
            <wsu:Created>2011-07-01T22:39:59.640Z</wsu:Created>
         </wsse:UsernameToken>
      </wsse:Security>
   </soapenv:Header>
   <soapenv:Body>
      <val:policyValuation>
         <inputXml><![CDATA[
         <Request>
                  <Parameters>
                           <Parameter NAME=”PolicyNumber”>VDA31012206</Parameter>
                           <Parameter NAME="PolicyValuesFlag">Yes</Parameter>
                           <Parameter NAME="Nearest">Yes</Parameter>
                           <Parameter NAME="EffectiveDate">11/02/2002</Parameter>
                  </Parameters>
         </Request>]]></inputXml>
      </val:policyValuation>
   </soapenv:Body>
</soapenv:Envelope>
```

FileReceived

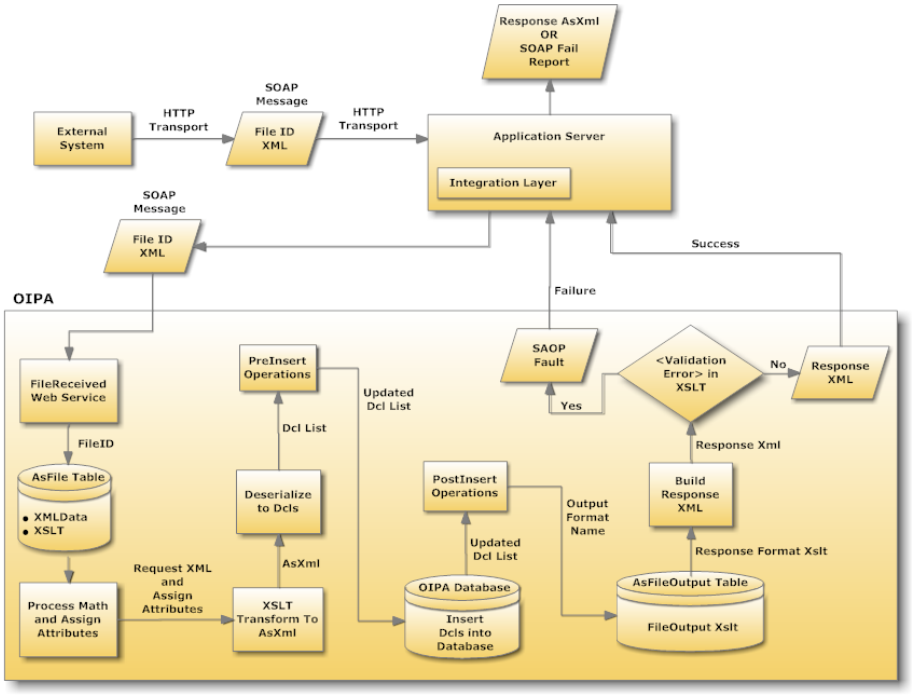
Introduction

The OIPA FileReceived Web Service allows an application to send data in XML format and execute core OIPA features against it. Based on the presence and functionality of extensions, this data can be modified, validated, and/or inserted in the OIPA database. A SOAP message is sent by a client application to the FileReceived Web Service. The SOAP message includes two parameters: FileID and XML. FileID identifies the configuration to use from the AsFile table for processing and transforming inbound XML into OIPA's AsXml. The XML element represents data to be sent to the OIPA for creating objects.

Once the SOAP message is received by the FileReceived Web Service, the AsFile entry is extracted from the database and the AsFile XMLData is processed (i.e., math is executed and attributes are assigned values). Data in the request XML is then transformed into AsXml using AsFile XSLT and assigned attributes.

A SOAP message is sent back to the requestor, or caller, including the result of the request. If the request was successful, then the message will consist of the transformed AsXml. This AsXml can also be transformed using XSLT picked from the AsFileOutput table indicated by Output AssignAttribute. If the request was not successful, then the Web Service response will consist of a SOAP Fault detailing the errors.

High Level Flow Overview



Inbound Flow

1. The FileReceived Web Service receives a request via a SOAP message.
2. The AsFile entry is looked up using the FileID specified in the request.
3. The math in the AsFile entry's XMLData is processed.
4. The AssignAttributes section in the AsFile entry's XMLData is processed.
5. The XSLT maps the request XML to AsXml.
6. The transformed AsXml is mapped to data objects.
7. PreInsert operations are performed on the objects.
8. Objects are inserted into the database.
9. PostInsert operations are performed on the objects.
10. Output XSLT from AsFileOutput is loaded based on assign attributes.
11. Response XML is built.
12. If the ValidationError section is configured in the XSLT, then a SOAP fault is created (with embedded response XML) and sent to the caller. Otherwise, response AsXml is returned.

Note:	Any of the above steps can be skipped if client extensions direct so. Additional functionality can also be performed at the beginning or end of every step.

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 a user-configurable portion of the FileReceived Web Service. This table contains a separate record for each type of file OIPA has been configured to receive. It has following columns:

- FileGUID – unique identifier for each entry in this table.
- CompanyGUID – unique identifier for a company.
- FileNameFormat – stores a descriptive name of the file format type.
- FileID – stores a unique three-character ID used to describe file format. An inbound SOAP message will include a <FileID> element specifying the format of the file to be used for processing.
- XMLData – specifies details about request type, math, assign attributes, pre-insert and post-insert.
- XSLT – XSLT for transforming inbound XML into AsXml.

XMLData

AsFile entry allows values to be assigned to various attributes before inbound XML undergoes the transformation process into AsXml. This is configured by using the <Attribute> element inside the <AssignAttributes> element. By assigning values at this stage of the process, pre-existing data from OIPA can be used to populate the AsXml.

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 of 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>
```

<RequestType>

This optional section specifies the type of operation like Insert or Quote. If this is missing, then the default request type is assumed to be of type Insert.

<Math>

This is an optional section that can run math on the values of the incoming XML. This section behaves the same as the Math section of a normal business rule. Math variables in this section will have a prefix specified by the ID attribute of the Math tag. These variables can then be used in the AssignAttributes section.

Example:

```
<File>
    <Math ID="Math">
        <MathVariables>
            <MathVariable VARIABLENAME="Number" TYPE="VALUE" DATATYPE="TEXT">
                31ALIC010801</MathVariable>
            <MathVariable VARIABLENAME="Prefix" TYPE="VALUE" DATATYPE="TEXT">AVA</MathVariable>
            <MathVariable VARIABLENAME="PolicyNumber" TYPE="EXPRESSION"
            DATATYPE="TEXT">Prefix+Number</MathVariable>
            <MathVariable VARIABLENAME="State" TYPE="VALUE"
            DATATYPE="TEXT">/NewPolicy/PolicyIssueState</MathVariable>
        </MathVariables>
    </Math>
    <AssignAttributes>
        <Attribute NAME="TestValue" TYPE="VALUE">Math:PolicyNumber</Attribute>
        <Attribute NAME="PolicyState" TYPE="XPATH">Math:State</Attribute>
```

```
   </AssignAttributes>
</File>
```

<AssignAttributes>

This is the parent element for one or more <Attribute> elements.

<Attribute>

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

Following is a list of available attribute 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 value of the NAME attribute 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 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 that are used for these operations. The architecture of the Pre- and Post-Insert functionality allows 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>
```

The following is a list of available Pre and Post Insert Processors:

Name	Parameters
AsFilePreInsertPercentInAllocationProcessorBll	N/A
AsFilePreInsertUvNavCalculationProcessorBll	N/A
AsFilePostInsertExposedComputationProcessorBll	ComputationID
AsFilePostInsertIndividualActivityProcessorBll	ActivityClientNumber
AsFilePostInsertProcessPolicy	PolicyNumber

<Output>

This is an optional element that allows the outbound AsXml to be manipulated by XSLT. It should have an attribute TYPE="TRANSFORM". The text of this element references an attribute from the Attributes section of the XMLData. The attribute should be the name of a record in the AsFileOutput table of the database.

AsXml

AsXml is the XML formatting used by OIPA to store data destined for the database. This format is very simple, using 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>
   <TableName>
      <ColumnName>Value</ColumnName>
   </TableName>
</AsXml>
```

Examples

An 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>
```

An 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>
```

XSLT

Overview

XSLT (Extensible Stylesheet Language) is an XML-based language used for the transformation of XML documents to other formats. Using XSLT, OIPA transforms the inbound payload of the SOAP message into AsXml, which can then be transformed into objects and 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, as well as to perform data validation and error handling.

All attributes that are used in the XSLT stylesheet must be defined either in the AsFile's XmlData column or in the Web Service's XML parameter. This is explained in the XSLT section.

Using Attributes from the XMLData Element of AsFile Entry

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, after the XSLT prolog, using the <xsl:param> tag.

For example, PolicyGuid, set in the XMLData section, can be referenced in the XSLT by declaring it in the beginning of the XSLT like this:

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

And then using it like this:

```
<xsl:element name="PolicyGuid">
   <xsl:value-of select="$PolicyGuid"></xsl:value-of>
</xsl:element>
```

Functions

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

- getNextGUID()
- getGmtTimeString()
- 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. For example, the following code will output a newly generated GUID inside the <PolicyGuid> element.

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

Functions can also be used to retrieve 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:getGmtTimeString (),
   $Offset))"/>
</xsl:template>
```

Validation and Error Handling

AsFile has the ability to perform data validations using the XSLT portion of the configuration. For 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, then the `<ValidationError>` block will be executed. As a result, a SOAP fault will be thrown, and the text within the element, ERRORSTATUSCODE and resulting XML will be returned to the caller.

Transformation Examples

Example 1:

XML Portion of SOAP Request

```
<NewPolicy>
   <PolicyName>TestPolicy</PolicyName>
</NewPolicy>
```

XSLT

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:helper="com.adminserver.webservice.helper.XsltFunctionHelper" extension-element-
prefixes="helper" version="2.0">
   <xsl:param name="SystemDate"></xsl:param>
   <xsl:template match="NewPolicy">
      <xsl:element name="AsXml">
         <xsl:variable name="PolicyName" select="./PolicyName"></xsl:variable>
         <xsl:element name="AsPolicy">
            <xsl:element name="PolicyName">
               <xsl:value-of select="PolicyName"></xsl:value-of>
            </xsl:element>
         </xsl:element>
      </xsl:element>
   </xsl:template>
</xsl:stylesheet>
```

Resulting AsXml After Transformation:

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

Example 2:

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>
```

Process Policy

- Allows invocation of processing for all pending activities on an existing policy.
- Takes as input XML that contains policy information identifying the policy to be processed.
- Returns a success indicator on successful processing; otherwise, returns a SOAP fault.

SOAP Request Example

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:proc="http://ProcessPolicy">
   <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>tester</wsse:Username>
            <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordText">tester</wsse:Password>
            <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary">FRkBB/REsT/4ThQzEjoiUQ==</wsse:Nonce>
            <wsu:Created>2011-07-01T22:39:59.640Z</wsu:Created>
         </wsse:UsernameToken>
      </wsse:Security>
   </soapenv:Header>
   <soapenv:Body>
      <proc:policyValuation>
         <inputXml><![CDATA[<Request>
            <Parameters>
               <Parameter NAME="PolicyNumber">VDA31022177</Parameter>
            </Parameters>
         </Request>]]></inputXml>
      </proc:policyValuation>
   </soapenv:Body>
</soapenv:Envelope>
```

Exposed Computation

Overview

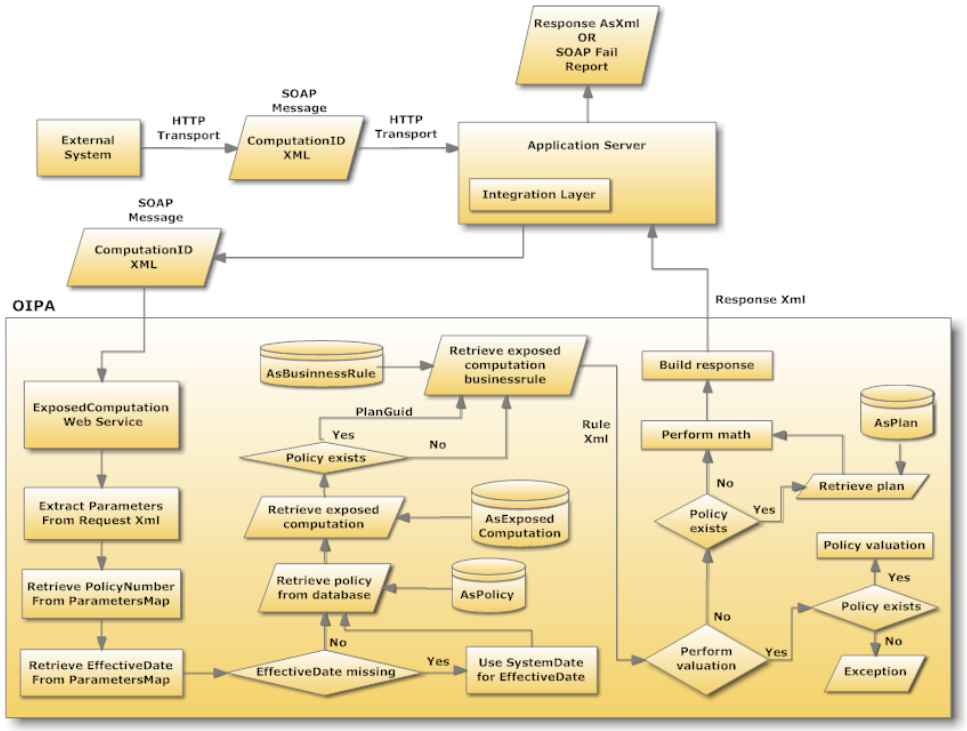
The Exposed Computation Web Service is exposed to give an external application access to OIPA's robust math engine. A call to this Web Service can be used to execute a calculation and return result values in the SOAP response XML.

A request to this service takes two text parameters:

- **ComputationID** – An identifier for a record in the AsExposedComputation table.
- **XML** – A text parameter that 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 to feed extra dynamic parameters to a request. Response from the call is also driven by the configuration of the ComputationID in the request.

Request Flow



Request Flow Diagram

The Exposed Computation Web Service processes a request with the following steps:

1. Parse policy number and effective date from the incoming XML document. Set policy context if policy information is supplied.
2. Load AsExposedComputation record for the ComputationID from the request.
3. Load ExposedComputation business rule (from AsBusinessRules table). If a policy is present, then load the business rule override for the plan of the policy.
4. Do policy valuation if a policy is present and exposed computation business rule is configured to do so.
5. Execute math from the Input configuration in the exposed computation business rule. If a policy is present, then use the data present in it and its plan.

6. Build response XML from the Output configuration in the exposed computation business rule AsExposedComputation Table

The AsExposedComputation table is a configuration table for storing exposed computations. An AsExposedComputation record will have a unique ComputationID value. The other relevant column 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 table for the exposed computation request is used to load an AsBusinessRule record that contains the configuration for what to process. The ExposedComputation business rule is set-up in much the same way as the Calculate business rules are configured for calculating segments. There is an Input element that contains the math variables configuration for processing the math engine. The math variables should be configured the same way as any other math section in the system. There is also an Output element, which contains the mappings for the input variables to output in the response. Below is a sample of a simple ExposedComputation 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, while the Output element pulls the 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 Parameter FIELD values when executing the math engine.

SOAP Request Example

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:exp="http://ExposedComputation">
   <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>tester</wsse:Username>
            <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordText">tester</wsse:Password>
            <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary">FRkBB/REsT/4ThQzEjoiUQ==</wsse:Nonce>
            <wsu:Created>2011-07-01T22:39:59.640Z</wsu:Created>
         </wsse:UsernameToken>
      </wsse:Security>
   </soapenv:Header>
   <soapenv:Body>
      <exp:processExposedComputation>
         <computationId>BC-Exp1</computationId>
         <inputXml><![CDATA[
```

```
         <Request>
            <Parameters>
               <Parameter NAME="PolicyNumber">VDA31011809</Parameter>
               <Parameter NAME=”EffectiveDate”>01/01/2009</Parameter>
               <Parameter NAME="InputVariable1">TestValue1</Parameter>
            </Parameters>
         </Request>]]></inputXml>
      </exp:processExposedComputation>
   </soapenv:Body>
</soapenv:Envelope>
```

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 perform valuation during the request.

EffectiveDate

An “EffectiveDate” parameter can be included 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 exposed computation 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 example, refer to 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>
```

DISQ

Overview

DISQ provides a simple, more comprehensive interaction model that requires no configuration. DISQ surfaces all of the data and associated operations in a consistent and flexible manner, facilitating any kind of service interaction.

While OIPA provides flexible services with AsFile and ExposedComputation, these services are difficult to understand, configure, and interact with. There is no formal XML schema that defines the input and output parameters for most web services. The payloads typically consist of a single string, which can be any kind of structured xml data.

In contrast, DISQ provides a formal XML schema which describes the data and operations that it exposes. DISQ requires no configuration in order for it to function, beyond normal configuration in the existing system. With a simple introduction to the DISQ model, one can easily understand how to make any request into OIPA.

Request Model

DISQ consists of a single Service Endpoint, with a flexible Request model which allows several operations to be invoked dynamically. A DISQ Request consists of the following parts:

1. **Header** - provides a loose bucket of properties that can help influence the overall processing of the DISQ request.
2. **Context** - identifies the resource that the DISQ request applies to. Context is a fundamental problem with OIPA because most times external systems do not know the OIPA unique identifier (i.e. GUID) of the resource they want to access. So the system requires a different method of "looking up" the resource to work with. The Context section of the DISQ request solves this problem by allowing the caller to specify any combination of fields to uniquely identify the context for the Action.
3. **Action** - identifies the operation to be performed within the context specified.
4. **ResponseFormat** - identifies the information to be returned as a result of the action being performed.

The contents of the request are dictated by the Action being executed. It is important to understand the usage models surrounding each Action in order to understand the appropriate Context and Response.

In the DISQ request sample above, we can see the following:

- The **Header** defines a *Message ID* and a *Timestamp*. These are supported in the standard DISQ Schema, and are typically used in SOA middleware.
- The **Context** defines the company that the client belongs to; in this case the caller is identifying the *Holding Company* as the context for this request.
- The **Action** being requested is to **Get** a Client with the last name *Hyatt*.
- The **ResponseFormat** is defined as *Default*. In the case of a **Get** action, the default response will be to return the fixed fields for the requested entity.

From a Query standpoint, it helps to read the request as follows:

Get me a Client with the Last Name 'Hyatt' that belongs to the Company named 'Holding Company'

Context

Context is one of the most important aspects to DISQ. Without holding the internal OIPA identifiers, external systems need some method of identifying the unique resource to use when executing an **Action**. The **Context** section is the DISQ means of identifying that resource.

Actions

Actions are the means of specifying the operation that is to be performed against the **Context**. The list of Actions that are available will grow overtime. At the time of this writing, the following Actions are expected:

- Get
- Find

Get Action denotes the retrieval of a single, unique resource in the system. The Get action will always return a single element. If more than one resource happens to be found, an exception will be thrown.

Find Action denotes the retrieval of multiple instances of the same source type, all instances of that type that match the search criteria will be found. **Find** is also different than **Get** in that it allows you to specify more than one data element as the action. When more than one element of the same name is specified, the result of the

data retrieval is to do a UNION of the results of processing each element individually. For example, if you want to find all clients with a last name of "Smith" who live in Boston OR Bedford, you could execute the following:

```
<Find>
  <Client>
    <LastName>Smith</LastName>
    <Addresses>
      <Address>
        <City>Boston</City>
      </Address>
      <Address>
        <City>Bedford</City>
      </Address>
    </Addresses>
  </Client>
</Find>
```

Response Format

The **ResponseFormat** section allows the caller to specify what to include in the **Response** from the **DISQ Request**. There are a few elements in the **ResponseFormat**:

ResponseType - can be Default, Acknowledge, or Custom. Default is the behavior if the Response Format element is omitted from the request.

- Default - the default response type is to return the subject of the action itself, and only the fixed fields. If dynamic fields are required, then the caller must specify a Custom response type.
- Acknowledge - a simple success or failure will be returned from the call
- Custom - the caller specifies what information is to be returned.

The example below is for a **Get Client** action. The **Custom** format indicates that the caller is requesting certain information to be returned. By specifying an empty **Fields** tag inside the **Client** tag, the caller is indicating to return dynamic fields. By specifying an empty **Addresses** tag inside the Client tag, the caller is indicating to return all addresses for the client.

```
<ResponseFormat>
    <ResponseType>Custom</ResponseType>
    <Client>
    <Fields/>
            <Addresses/>
    </Client>
</ResponseFormat>
```

Responses

The **Responses** are returned from the execution of a **DISQ Request** can contain any OIPA type defined in the oipa schema. Typically, these will be the entities that are the subject of the action being requested. For example, for **Get Client**, a **Client** element will be returned. This section highlights an example response.

Example: **Get Client** request with a **Default** response type.

DISQ Request

```
<DisqRequest>
   <Header>
      <MessageId>671B973D-2AC7-4572-8E23-202FF4C5AB64</MessageId>
      <Timestamp>2012-09-19T17:00:30.496-04:00</Timestamp>
   </Header>
   <Context>
      <Company>
         <CompanyName>Holding Company</CompanyName>
      </Company>
   </Context>
   <Get>
      <Client>
         <ClientGuid>0872C75B-B50F-47E1-B439-BA3DC3BBF88A</ClientGuid>
      </Client>
   </Get>
   <ResponseFormat>
      <ResponseType>Default</ResponseType>
   </ResponseFormat>
</DisqRequest>
```

DISQ Response

```
<Client>
   <FirstName>Deb</FirstName>
   <LastName>Hyatt</LastName>
   <TaxId>123412345</TaxId>
   <ClientGuid>0872C75B-B50F-47E1-B439-BA3DC3BBF88A</ClientGuid>
   <DateOfBirth>1976-01-01T00:00:00-05:00</DateOfBirth>
   <UpdatedGmt>2009-06-02T16:49:10-04:00</UpdatedGmt>
   <MiddleInitial></MiddleInitial>
   <Sex>02</Sex>
   <TypeCode>02</TypeCode>
   <LegalResidenceCountryCode>US</LegalResidenceCountryCode>
   <MaritalStatus>00</MaritalStatus>
   <EntityTypeCode>CLIENT</EntityTypeCode>
</Client>
```

OUTBOUND WEB SERVICES

Outbound Services Connector (OSC)

The OSC leverages the extensibility framework to allow for outbound web service calls during math processing. The extension provides mechanisms to specify delivery targets and message templates. The latter allows users to map between the data at a given extension point and a downstream interface. Further, it avoids the need to develop individual extensions for each downstream interface, while maintaining application performance and throughput.

Note: A setup and configuration guide for the OSC is provided in the 9.7.1.0.0 documentation release library on OTN.

Deleted: 9.7.0.0

FILE RECEIVED EXAMPLE USING ACORD 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 ACORD 103 implementation, which is included in version 9.1 of OIPA.

SOAP Request

The SOAP request message must include an element indicating the service to use, which is 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 the system 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 that may be misinterpreted, such as "<" and "&", must be replaced with their respective escape sequences ("`<`" and "`&`" in this case).

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:fil="http://FileReceived">
   <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>tester</wsse:Username>
            <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
            username-token-profile-1.0#PasswordText">tester</wsse:Password>
            <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
            wss-soap-message-security-1.0#Base64Binary">kC5eI6iq8x17/qA3mzs6/g==</wsse:Nonce>
            <wsu:Created>2010-03-22T14:12:34.223Z</wsu:Created>
         </wsse:UsernameToken>
```

```
        </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
        <fil:processFileReceived>
            <fileId xsi:type="xsd:string">AL2</fileId>
            <xml xsi:type="xsd:string">
                &lt;NewField&gt;
                    &lt;PlanName&gt;Functional Prototype Plan&lt;/PlanName&gt;
                    &lt;FundName1&gt;Interest Rate Calc Fund 1&lt;/FundName1&gt;
                    &lt;FundName2&gt;Interest Rate Calc Fund 2&lt;/FundName2&gt;
                    &lt;FundName3&gt;Interest Rate Calc Fund 3&lt;/FundName3&gt;
                    &lt;FundName4&gt;Interest Rate Calc Fund 4&lt;/FundName4&gt;
                &lt;/NewField&gt;
            </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. In this example, 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: success or a fault.

Successful Response

When an inbound SOAP request is successfully processed, a SOAP response is returned to the caller along with the transformed AsXml that was inserted into the database. This default configuration can be changed and the data returned to the caller can be modified by inserting an XSLT stylesheet into 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>
        <ns1:processFileReceivedResponse
        soapenv:encodingStyle=""http://schemas.xmlsoap.org/soap/encoding/""
        xmlns:ns1=""http://FileReceived"">
            <processFileReceivedReturn xsi:type=""xsd:string"">
            <![CDATA[<?xml version=""1.0"" encoding=""UTF-8""?>
                <AsXml>
                    <AsAllocation>
                        <AllocationGuid>09B57F68-0B2D-476D-AA4F-806CBA5D118A</AllocationGuid>
                        <Value>100</Value>
                        <GroupGuid>9FF5FE2A-D5EC-4B77-A5F5-DF44E6AB0BC7</GroupGuid>
                        <TypeCode>03</TypeCode>
                        <RelatedGuid>A3CCC022-A7A3-1BAD-E040-8C0A0EA651AF</RelatedGuid>
                        <FundGuid>66E7A284-2959-4261-84E5-FB81900AC504</FundGuid>
                        <AllocationMethodCode>01</AllocationMethodCode>
                        <AllocationPercent>20</AllocationPercent>
                        <AllocationAmount>10</AllocationAmount>
                        <AllocationUnits>2</AllocationUnits>
                        <PercentInAllocation>0</PercentInAllocation>
                        <EffectiveDate/>
                    </AsAllocation>
                    <AsAllocation>
                        <AllocationGuid>717A1C87-AFFF-4BF1-86CD-F525A941D29B</AllocationGuid>
                        <Value>100</Value>
                        <GroupGuid>9FF5FE2A-D5EC-4B77-A5F5-DF44E6AB0BC7</GroupGuid>
                        <TypeCode>03</TypeCode>
                        <RelatedGuid>A3CCC022-A7A3-1BAD-E040-8C0A0EA651AF</RelatedGuid>
                        <FundGuid>56FE02B5-E24F-41CB-BBAB-FD5A1212210F</FundGuid>
                        <AllocationMethodCode>01</AllocationMethodCode>
                        <AllocationPercent>20</AllocationPercent>
                        <AllocationAmount>10</AllocationAmount>
                        <AllocationUnits>2</AllocationUnits>
                        <EffectiveDate/>
                    </AsAllocation>
                    <AsAllocation>
                        <AllocationGuid>914C2901-6838-495E-BF81-01DA9423E624</AllocationGuid>
                        <Value>100</Value>
                        <GroupGuid>98DEFD38-40EA-4562-A023-CD80F77936E8</GroupGuid>
                        <TypeCode>03</TypeCode>
                        <RelatedGuid>A3CCC022-A7A3-1BAD-E040-8C0A0EA651AF</RelatedGuid>
                        <FundGuid>E5840467-A35B-44CF-8FAD-3CB03AFB2F3A</FundGuid>
                        <AllocationMethodCode>01</AllocationMethodCode>
                        <AllocationPercent>20</AllocationPercent>
```

```
                        <AllocationAmount>10</AllocationAmount>
                        <AllocationUnits>2</AllocationUnits>
                        <PercentInAllocation>0</PercentInAllocation>
                        <EffectiveDate/>
                     </AsAllocation>
                     <AsAllocation>
                        <AllocationGuid>305A4E2E-E3B4-4756-B48A-B7399AB564E2</AllocationGuid>
                        <Value>100</Value>
                        <GroupGuid>98DEFD38-40EA-4562-A023-CD80F77936E8</GroupGuid>
                        <TypeCode>03</TypeCode>
                        <RelatedGuid>A3CCC022-A7A3-1BAD-E040-8C0A0EA651AF</RelatedGuid>
                        <FundGuid>8DAE3947-83F6-4189-BB3F-DCA40F3AAAC9</FundGuid>
                        <AllocationMethodCode>01</AllocationMethodCode>
                        <AllocationPercent>20</AllocationPercent>
                        <AllocationAmount>10</AllocationAmount>
                        <AllocationUnits>2</AllocationUnits>
                        <PercentInAllocation>0</PercentInAllocation>
                        <EffectiveDate/>
                     </AsAllocation>
                  </AsXml>]]>
               </processFileReceivedReturn>
            </ns1:processFileReceivedResponse>
         </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 details 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:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <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>
```