

Oracle® Insurance Policy Administration

Web Services

Version 9.3.1.0

Documentation Part Number: E21044_01

May 2011

Copyright © 2009, 2011, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Where an Oracle offering includes third party content or software, we may be required to include related notices. For information on third party notices and the software and related documentation in connection with which they need to be included, please contact the attorney from the Development and Strategic Initiatives Legal Group that supports the development team for the Oracle offering. Contact information can be found on the Attorney Contact Chart.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your beta trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

Table of Contents

Web Services Overview	5
Creating Messages with SOAP	6
SOAP Overview	6
WSDL	6
Security	7
OIPA Web Services	8
PAS Properties	8
ValuePolicy	8
FileReceived	9
Introduction	9
High Level Flow Overview	10
AsFile Overview	11
XSLT	14
Functions	15
Validation and Error Handling.....	15
A Transformation Example	16
AsXml	17
Schema.....	17
Examples	17
Exposed Computation	18
Overview	18
Request Flow	19
AsExposedComputation Table	20
Exposed Computation Business Rule	20
Valuation	21
Exposed Computation SOAP Messages.....	21
Additional File Received Examples using Acord LOMA.....	23
SOAP Request	23
Full Request.....	23
XML Message.....	24
SOAP Response.....	25

Successful Response	25
SOAP Fault.....	25
XMLData	26
XSLT	26
Format	26
Templates	27
Data Validation	28

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.
- [RemoteCompilation](#) – Service for compiling a transaction that is received. It will return any compilation errors that are generated.
- [ValuePolicy](#) – Service for running valuations on a policy.

This document not only discusses the available Web Services, but also gives basic overviews 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.

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. 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. 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 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 WSDL can be found here:

<http://<OIPA-SERVER:9080>/PASJava/service/> (for JBoss)

<http://<OIPA-SERVER:8080>/pas.web/service> (for Tomcat)

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) 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#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. This is mainly used while setting up the Palette.

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>
```

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>
```


FILERECEIVED

Introduction

The OIPA FileReceived Web Service allows an application to send data in XML format and run application 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, AsFile entry is extracted from database and 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.

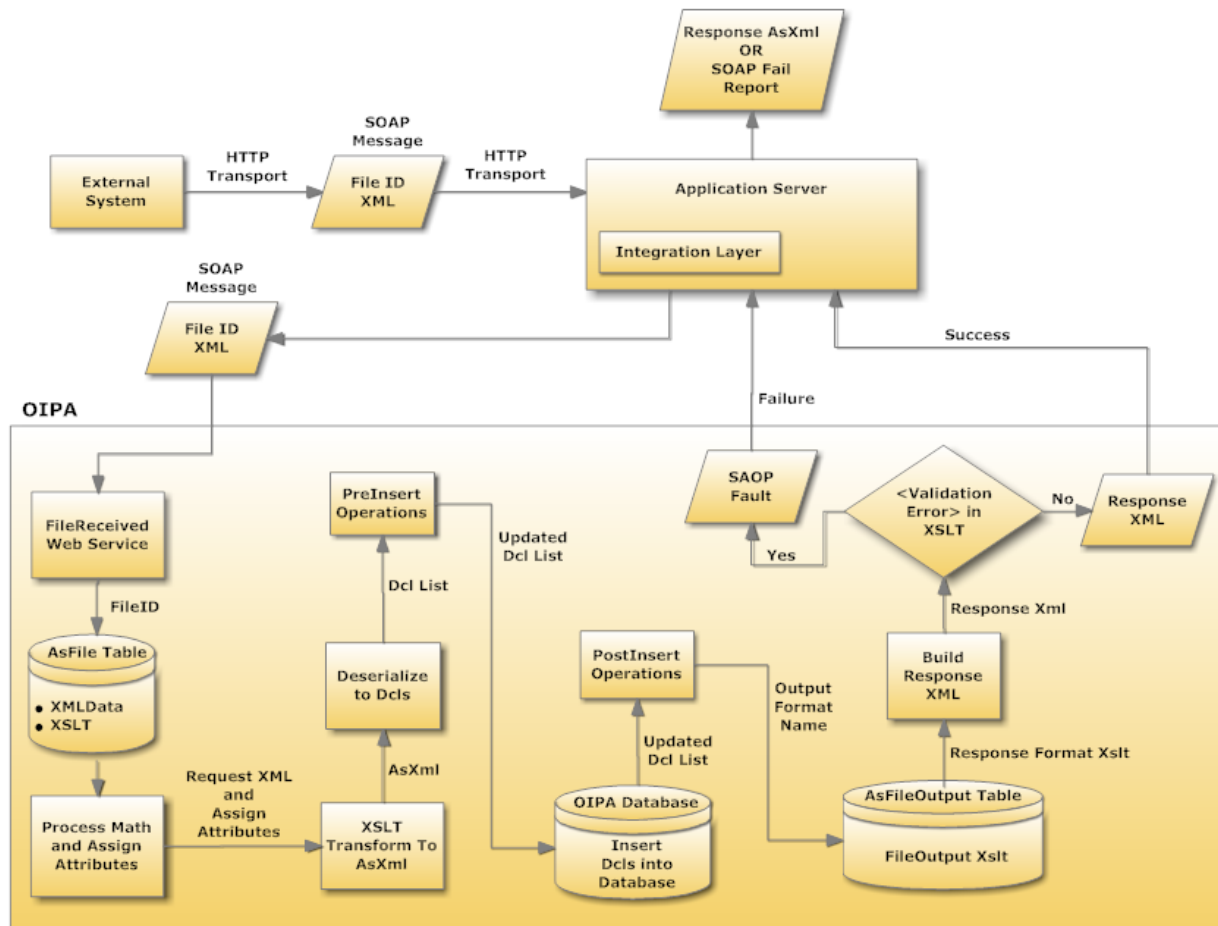
A SOAP message is sent back to the requestor, or caller, including the result of the request. If the request was successful, the message will consist of the transformed AsXml. This AsXml can also be transformed using XSLT picked from the AsFileOutput table indicated by Output AssignAttribute. If the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

The WSDL for the FileReceived service is located here:

<http://<OIPA-SERVER:9080>/PASJava/service/FileReceived?wsdl> (for JBoss)

<http://<OIPA-SERVER:8080>/pas.web/service/FileReceived?wsdl> (for Tomcat)

High Level Flow Overview



High Level Flow Diagram

Inbound Flow

1. FileReceived Web Service receives a request via a SOAP message.
2. AsFile entry is looked up using FileID specified in the request.
3. Math in AsFile entry's XMLData is processed.
4. AssignAttributes in AsFile entry's XMLData are 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. Load output XSLT from AsFileOutput based on assign attributes.
11. Build response Xml.
12. If Validation Error section is configured in the XSLT then 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 in 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 <Attribute> the 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.

<RequestType>

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

<Math>

This is an optional section that can run math on the values in the incoming XML. This 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 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 <code>asc_NextSequenceInteger</code> 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 other system functionality 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>
```

<Output>

This is an optional element that allows the outbound AsXml to be manipulated. 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.

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 and 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 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 functionality 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. 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:getGmtTime(),
    $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 `<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.

A Transformation Example

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>
```


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

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

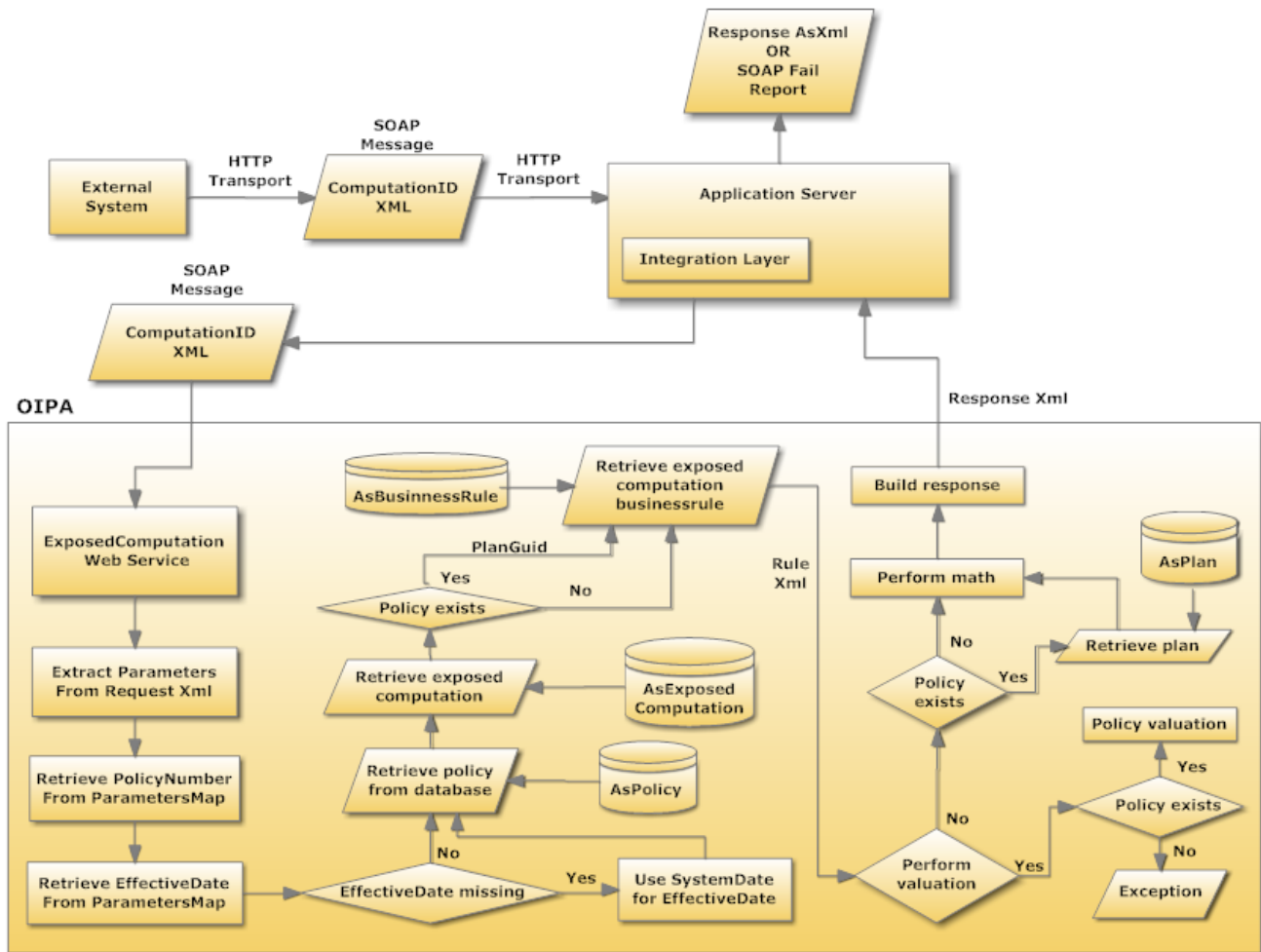
The Exposed Computation Web Service is exposed to give an external application access to OIPA's robust math engine. A call to this Web Service can be used to execute a calculation and return result values in the SOAP response XML. The Exposed Computation Web Service is available by using "processExposedComputation" operation of the Exposed Computation Web Service and WSDL 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 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 exposed computation business rule (from AsBusinessRules table). If a policy is present then load 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 processing 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 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 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 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 Parameter 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 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>
```

ADDITIONAL FILE RECEIVED EXAMPLES 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.

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 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: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. 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 a 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>
    <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 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: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 a rule to be set for use 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:template>
```

Data Validation

The XSLT style sheet can be used to test information in the inbound request and make any changes necessary to be compatible 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>
```