ORACLE
INSURANCE

Oracle® Insurance Policy Administration

Activity Processing

Version 10.1.1.0

Document Part Number: E55503-01

August, 2014

Copyright © 2009, 2014, Oracle and/or its affiliates. All rights reserved.

Trademark Notice
Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

License Restrictions

Warranty/Consequential Damages Disclaimer
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.

Warranty Disclaimer
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.

Restricted Rights Notice
If this is software or related documentation that 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 END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

Hazardous Applications Notice
This software or hardware 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 or hardware in dangerous applications, then you shall be responsible to take all appropriate failsafe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Third Party Content, Products, and Services Disclaimer
This software or hardware 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.

Table of Contents

INTRODUCTION

Activity processing is a core component of the Oracle Insurance Policy Administration (OIPA) system. Every administrative event that occurs in an insurance policy, plan, client or company is described as an activity in the system. The purpose of this guide is to provide a comprehensive explanation of activity processing in OIPA.

CUSTOMER SUPPORT

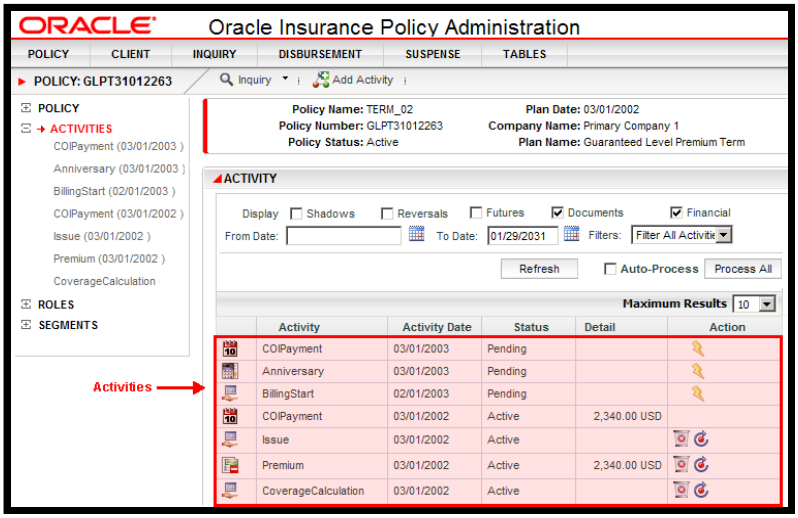
If you have any questions about the installation or use of our products, please visit the My Oracle Support website: https://support.oracle.com, or call (800) 223-1711.

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.

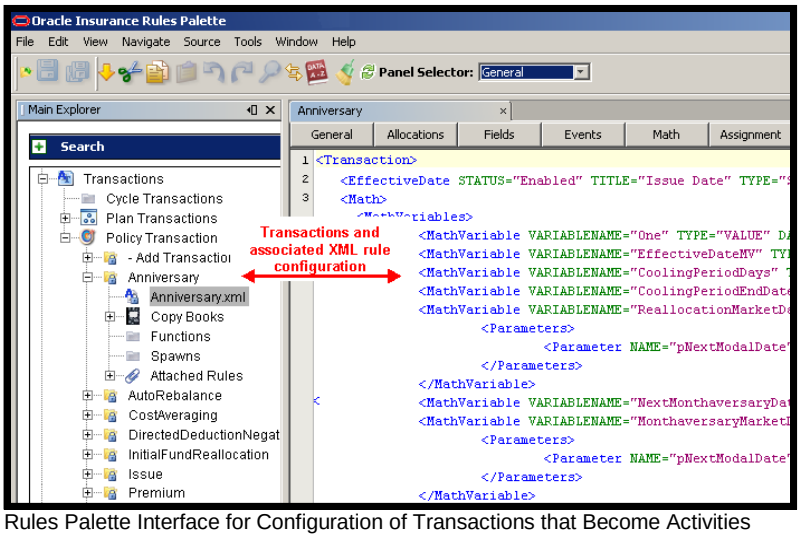
OVERVIEW

An activity is an event that may or may not change relative data based on a business's requirements.

An activity is an instance of a transaction. A transaction can be considered as synonymous to a class and an activity is synonymous to the object that is an instance of that class. Transactions are XML rules that are configured according to business requirements. They are configured using the Oracle Insurance Rules Palette. Transactions define input variables, processing logic and lists of changes made to data such as policy information. Transactions can be defined in the system at the policy, plan, client and company level depending on the type of data they need to execute. Some typical OIPA transactions at the policy level are premium, billing and anniversary processing.



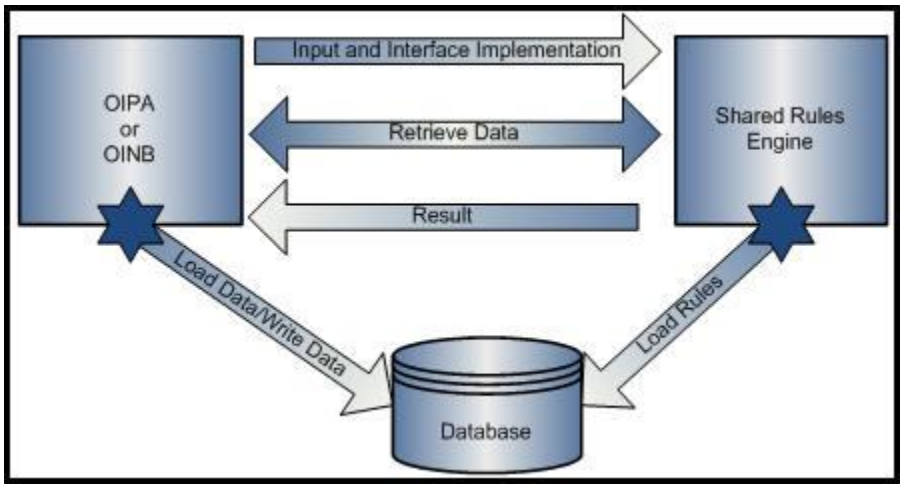
Front-end OIPA User Interface Listing Activities that Occurred at the Policy Level



Rules Palette Interface for Configuration of Transactions that Become Activities

SHARED RULES ENGINE

The Shared Rules Engine (SRE) component performs activity processing for the OIPA application. Activity processing manages insurance events. SRE loads a transaction and processes the data according to the business rules and math associated with the transaction. When processing completes, the results are sent back to the calling application. The database stores both the configured transaction rules and the actual insurance data.



High Level Interaction Diagram

The above diagram shows a high level interaction between the calling application and SRE. The calling application calls SRE and provides input data and an interface to callback the calling application for extra data needed. SRE does not directly make calls to the database, except for loading rules attached to the current activity. SRE loads the transaction and retrieves any other rules associated with the transaction. When processing is complete, the results are packaged and returned to the calling application and then the results are committed to the database.

There are five components of SRE that come together in processing an activity. They are as follows:

1. Processor
2. Generators
3. Math
4. Application Process Execution (Part of the calling application, but SRE calls into the application process execution during activity processing.)
5. Extensibility

INTERFACE BETWEEN THE SHARED RULES ENGINE AND OIPA

This section describes how SRE and the calling application communicate. Currently SRE and the calling application are not completely separated with interfaces. The calling application directly calls SRE to start processing and SRE libraries are required for the calling application to be compiled.

To begin activity processing, the calling application calls a process method in SRE's class; ActivityProcessorBll. The process method has three input parameters and returns an ActivityProcessResultsDcl.

Input Parameters

1. **VariableHashMap –** A collection of key-value pairs. The key is a string and the value is an instance of VariableDcl. Data is flattened into a key-value pair for lookup.
   - **Activity:FieldsXX** would be used for activity data.
   - **Policy:FieldXX** would be used for policy data.
   - **Transaction:XXXX** would be used for transaction data
   - **Plan:XXXX** would be used for plan data
   - **Product:XXXX would be used for product data**
   - **Client:XXXX** would be used for client data
   - **Company:XXXX** would be used for company data
   - **Address:XXXX** would be used for address data
   - **Program:XXXX** would be used for program data
   - **Withholding:XXXX** would be used for withholding data
   - **AllocationFromFundGuidArray** would be used for From allocation data
   - **AllocationToFundGuidArray** would be used for To allocation data

2. **IApplicationCallback –** An "umbrella" interface for all callback interfaces. The callback interfaces are as follows:
   - **DataRetriever** – Executes SQL statements and named queries related to activity processing. Results from the database are returned to SRE as DataDcls, which contain row and column details from the result set.
   - **RateRetriever** – Retrieves rates depending on the rate description and the criteria for the rates. The calling application can store the rates for insurance in any manner and implement this interface for processing needs specified in the rules.
   - **IActivityTaskExecutor** – Processes other activities as part of this parent activity. This is used especially when running backdated activities. To process backdated activities, all active activities that appear in the activity timeline after the backdated activity must be undone and then the backdated activity must be processed. This interface is used to run other activities and commit them as part of the outer processing activity.
   - **IPolicyValuationBll** – Values a policy and returns the cash value. It also is used to locate details about the funds and their cash value, as well as deposits and removal history.
   - **ICurrencyBll** – This interface is used to load currency and round currency information.

- **IActivityFunctionFactory –** This interface is used to create activity functions.
- **IAddressCallbackBll -** This is the callback interface for the AddressScreen rule.
- **IClientCallbackBll -** This is the callback interface for Client details.
- **IDataFieldRetriever –** Retrieves or updates a value or object from the domain object model using the given xpath expression.

3. **ActivityProcessType** – An enumerated type (enum) that specifies the type of activity processing logic.

Output Result

ActivityProcessResultDcl – A complex Dcl that contains the inputs passed, math calculation variables, errors if applicable and a list of updates, inserts and deletes to the data as part of the rule processing. This data is then iterated to be updated to the database.

UNDERSTANDING ACTIVITIES

Activity processing is controlled by various attributes associated with the activity. Activities are instances of XML transaction rules being applied on data at a specific level in the application. The AsActivity table stores records for activity processing that house applicable business event data. The AsTransaction table stores the XML transaction logic that processes activity data. There are three important areas to focus on when discussing activities: transaction type, activity type and activity status. Each of these areas is tracked using code values. These codes may be found in the AsCodes table or from **Admin Explorer | Codes** in the Rules Palette. The code values used in activity processing are used by the system and should not be changed.

TRANSACTION TYPES

The transaction type code, which is stored in AsTransaction in the XML transaction rule associated with the activity, plays an important role in activity processing. The transaction type code specifies the type of data or the level where the activity will process. This then drives the type of processing, such as math or valuation, that should be executed by the system. The type code definition can be located in the AsCode table under AsCodeTransactionType.

OIPA Transaction Types

Transaction Type	Description
Policy Financial	This transaction executes at the policy level and may or may not have financial impact. It participates in undo/redo and can be recycled from the user interface.
Policy Financial Non Reversible Non Reversing	This transaction executes at the policy level and may or may not have financial impact. It does not participate in undo/redo and cannot be recycled via the user interface.
Policy Document	This transaction executes at the policy level and generates documents or reports. It participates in undo/redo and can be recycled from the user interface.
Policy Document Non Reversible Non Reversing	This transaction executes at the policy level and generates documents or reports. It does not participate in undo/redo and cannot be recycled from the user interface.
Policy Financial Reversible Non Reversing	This transaction executes at the policy level and may or may not have financial impact. It participates **partially*** in undo/redo and can be recycled from the user interface.
Policy Document Reversible Non Reversing	This transaction executes at the policy level and generates documents or reports. It participates **partially*** in undo/redo and can be recycled from the user interface.
Plan Financial	This transaction executes at the plan level and may or may not have financial impact. It does not participate in undo/redo and can be recycled via the user interface.
Plan Financial Non Reversible Non Reversing	This transaction executes at the plan level and may or may not have financial impact. It does not participate in undo/redo and cannot be recycled via the user interface..
Plan Document	This transaction executes at the plan level and generates documents or reports. It does not participate in undo/redo and can be recycled via the user interface.
Plan Document Non Reversible Non Reversing	This transaction executes at the plan level and generates documents or reports. It does not participate in undo/redo and cannot be recycled via the user interface.
Client Financial	This transaction executes at the client level and may or may not have financial impact. It participates in undo/redo and can be recycled from the user interface.

Client Financial Non Reversible Non Reversing	This transaction executes at the client level and may or may not have financial impact. It does not participate in undo/redo and cannot be recycled via the user interface.
Client Document	This transaction executes at the client level and generates documents or reports. It does not participate in undo/redo and can be recycled via the user interface.
Client Document Non Reversible Non Reversing	This transaction executes at the client level and generates documents or reports. It does not participate in undo/redo and cannot be recycled via the user interface.
Client Financial Reversible Non Reversing	This transaction executes at the client level and may or may not have financial impact. It does not participate in undo/redo and can be recycled via the user interface.

***Partially**: May be inserted between existing active activities without invoking undo-redo on any of the processed activities after it. Once processed, will be recycled/reprocessed due to the reprocessing of any "Policy Financial" or "Policy Document" activities prior to it.

ACTIVITY TYPES

Each activity record has an activity type code that is stored in the AsActivity table. The type code definition can be found in the AsCode table under AsCodeActivityType. These types should not be confused with the transaction type code or the status of an activity, but instead, used in conjunction with them to understand how an activity was generated and what status the activity is currently in. Activities can be generated by an end user or the system may automatically generate activities because a dependent activity's data was changed.

OIPA Activity Types

Code Name	Code Value	Description
Natural	01	Activity entered manually by a user. Activity that was spawned for the first time from a natural activity. A spawned activity even though system generated can be considered a natural activity because the user manually processed the activity that spawned it. Activity created by a web service.
Reversal	02	Reversal activity that was created by an end user when either manually deleting or recycling an activity. Spawned activity that was reversed because the originating activity was manually reversed.
Undo	03	Activity that is created to reverse an active activity that is created by the system as part of running another reversal or as part of processing a pre-dated activity. This behaves exactly as the Reversal but just differentiates itself as system generated.
Redo	04	System generated activity that was automatically created due to the generation of an Undo activity.
Deleted	05	Currently not used.

ACTIVITY STATUS

The activity statuses are fundamental for activity processing and historical recording. They indicate at the activity level the status of that activity record. In comparison, the activity types section records the type of activity that was processed. The activity status, with the date stamp in current and history records, identifies the significant point of process and provides internal control for activities.

Status Code

Code Name	Code Value	Description
Pending	02	The activity is not yet processed. Pending data requires action before applying to current processing and math calculation. All required data must be entered and the activity processed to change the status from pending to active.
Active	01	Indicates the activity is active. Refers to current data that has completed activity processing and math calculation. This includes processing, any changes to table and inserting XML to write to the table. No more processing can be done on this activity.
Pending Ready	09	An attempt was made to run the activity but was unsuccessful.
NUV Pending	13	The activity is active but it does not have NUVs for some or all of the funds associated with activity processing. This will process later when NUVs become available. This status does not invoke undo/ redo processing for future active activities.
Gain Loss Pending	14	This activity is active but gain loss calculation is pending and is not complete. This will be processed later when NUVs are available. This status does not invoke undo/ redo processing for future active activities.
Shadow	12	This activity is effectively deleted from the system from an end user perspective as it is a result of an activity being reversed. It is available in the database and system for auditing purposes.
Pending Shadow	34	An activity whose data was entered, but never processed and then deleted.
Requirement Pending	57	An activity that has pending activity level requirements has this status.
Queued	58	If the Transitions/Queue configuration is present and there are prior activities in NUV Pending status that share allocation funds with the current activity, the activity will go into Queued status when either a user or nightly cycle attempts to process the activity. It will remain in this status until there are no prior activities in NUV Pending status that share common funds in the activity allocations.
Enrollment Pending	59	An Enrollment activity that is pending.
Activity Sequence Pending	70	A sequence activity that is pending.
Queued Activity Sequence	71	A sequence activity that has been queued.
Activity Sequence Error	72	A sequence activity that encountered an error during processing.
Processing Wait	97	This status implies that the activity is executing a long running task and is waiting for that task to complete.
Processing Stopped	98	This status implies that the activity has stopped processing a long running task and has ended in an error.

Sub Status Code

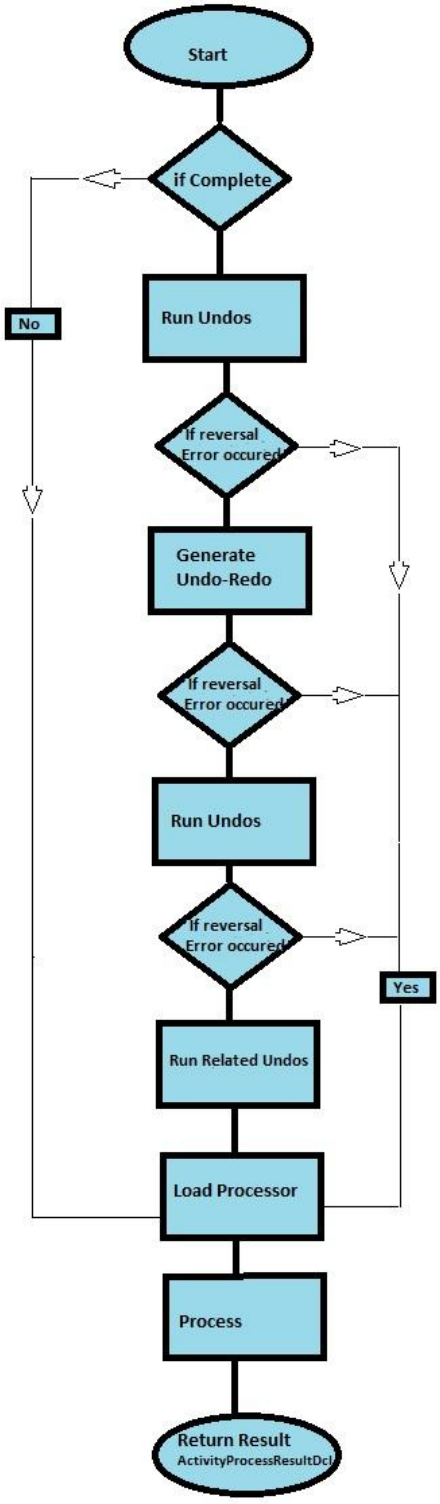
Code Name	Code Value	Description
Cancelled	01	Only applies to policy financial activities in Status Code 12 and 34. Activities in this sub status code are visible by default on the policy activity screen activity grid and are displayed as Cancelled. Regular activities in Status Code 12 and 34 without a sub status code are not displayed by default on the policy activity screen activity grid.

Error Status Code

Code Name	Code Value	Description
No Errors	01	Activity did not generate any business errors.
Business Error	02	Activity generated business errors.

Activity Processing Flow

The activity processing flowchart reveals the system steps.



Start – The shared rules engine receives a request from OIPA. The processing proceeds only if the activity is not active.

If "Complete" – If standard activity processing should be invoked. Value of ***ActivityProcessType.***
It is sent by OIPA. It has three possible values: COMPLETE,

SKIP_UNDOREDO_GENERATION or QUOTE. Strip down processing is done for options other than COMPLETE. QUOTE is for quoting an activity. SKIP_UNDOREDO_GENERATION for non-reversing activities, undo processing and for a specific instance during cycle processing after processing one activity in a policy. COMPLETE is the default option.

Run Undos – This step looks for all pending undo activities that need to be run with an effective date after the current activity effective date and executes them. This logic calls back into OIPA and it calls the shared rules engine in recursion to execute the undo activity. If there are no activities in future relative to the current then this step is skipped.

Generate Undo/Redo – This step looks for all activities that are active with an effective date after the current activity effective date. It creates an Undo/Redo for those activities.

Move ready to pending – This looks for activities in PendingReady (09) status on or before the current activity, and moves them to Pending (02) status.

Run Undos – If in the previous steps there are any activities generated then this step runs the undos of the activities generated.

Run related undo – This step checks if this activity is created by recycle and corresponding UNDO is still pending then execute here to make sure it is executed as the last before processWithoutUndoRedoGeneration

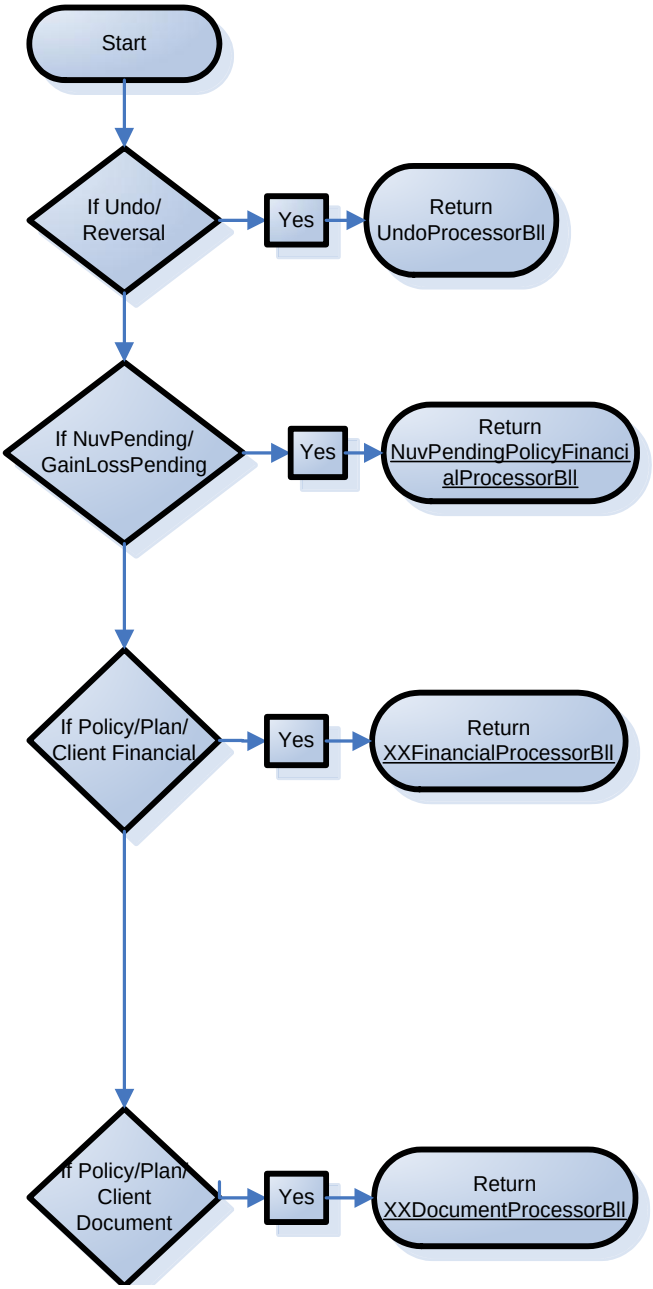
Load Processor – This loads the corresponding processor depending on the activity type code, activity status code and transaction type code.

Process - Call the processor process method. This is explained in detail in the next section of this document.

Business error on reversal – If an error occurs during activity reversal, the further processing stops and error is thrown.

SUBCOMPONENTS OF THE SHARED RULES ENGINE

Depending on the activity type that is sent for processing, an appropriate processor is initiated that handles the processing steps. The different processing types and diagram are as follows:



Undo/Reversal Activity – Activity is already processed and it needs to be undone. Handled by UndoProcessorBll.java.

Nuv Pending/Gain Loss Pending Activity – Activity is processed and is in active status, but some NUV's are missing or Gain Loss calculation is missing due to missing data. Handled by NuvPendingPolicyFinancialProcessorBll.java.

Policy Level Activity – Activity at the policy holder that impacts the policy alone. Handled by PolicyFinancialProcessorBll.java.

Client Level Activity – Activity at the client level that impacts client data and might impact all policies the client holds. Handled By ClientFinancialProcessorBll.java.

Plan Level Activity – Activity at the plan level that aggregates all policies in the plan like reports or other changes to the plan. Handled by PlanFinancialProcessorBll.java.

Document Generation – Activities that generate only reports are handled by DocumentProcessorBll classes. Separate classes exist for Policy Level Documents (PolicyDocumentProcessorBll), Plan level documents (PlanDocumentProcessorBll) and Client level Documents (ClientDocumentProcessorBll).

LIST OF PROCESSES AS PART OF ACTIVITY PROCESSING

Depending on the processors, different sections of activity processing are executed. The processes are as follows:

- **doPreliminaryForForward** – Checks the transaction's eligibility for processing and loads NUVs for funds and prepares the activity for processing.
- **doSuspense** – Processes suspense for funds received.
- **doValuation** – Values the policy of all funds and calculates the cash value and other variables. This is called only when the transaction calls for the valuation in its rules. SRE calls the calling application to do the valuation using the interface. The calculated values are later used in other sections of activity processing.
- **doMath** – Calculates the math section of rules.
- **doBusinessLogic** – Runs the application process execution associated with the activity.
- **doAssignment** – Runs assignment processing.
- **doTransition** - Processes Transition element in Transaction Xml. If required, moves activity to Queued status and modifies its effective date.
- **doPostAssignmentValidation -** Processes the PostAssignmentValidateExpressions rule overridden at the Transaction level thus enabling validation and creation of business errors after Assignment processing.
- **doDisbursement** – Runs disbursement processing.
- **doAccounting** – Runs accounting for bookkeeping purposes.
- **doSpawn** – Runs spawn logic to spawn new activities based on the transaction's specific rules.
- **doActivitySequence** – Processed activities as per the sequence specified.

There are various sub processes that run during activity processing. Processors like Undo and NuvPending run a few of these and also run other processes, such as loading the changes that happened during activity processing and reversing those changes.

GENERATORS

Generators are classes that produce other classes for execution. XML rules are configurable with expressions and conditions, and generators are used to execute these rules. Generators are responsible for loading the rules, performing error checking, translating the rules and creating java source code at run time for the rules then compiling them into classes. They also create an instance of the run time generated class and return them to the caller for execution.

Generators have the following functions:

- Load rules.
- Parse rules and check for errors. Report Errors if needed.
- Translate rules to java code and compile the class.
- Cache the translation for next time lookup.

Generators run in the modes described below. The mode can be set in the application property file, such as PAS.properties. Information regarding the PAS.properties file can be located on Oracle's Technology Network in the OIPA 10.1.0.0 Documentation Library E55027-01.

The PAS.properties file section where you set the application mode:

#--
 # application mode (DEVELOPMENT or PRODUCTION)
 # Development mode is where configuration changes are allowed.
 # Production mode is where configuration change is a new release and JVM is restarted when they are changed.
#--

application.mode= DEVELOPMENT

In DEVELOPMENT mode the system allows rules to be changed in the database during application runtime. This mode should only be used during active development. Generators load the rules every time, generate a hash key and cache the generated classes associated with the hash key. If the rules are changed using Oracle Insurance Rules Palette, then the hash key generated will be different, which will force the generator to translate and compile again. If the rules are not changed then it reaches out to its cache and returns the cached instance.

application.mode= PRODUCTION

In PRODUCTION mode the system does not allow the changing of rules. If rules are modified, it requires that all JVMs be stopped and restarted so that caches are cleared. In production mode, rules are cached as well as the translated classes. Hence no check is made to ensure changes.

Generators support debugging mode and non-debugging mode. The Rules Palette can debug into transactions and do a line by line execution of the math section using a web service. In order to debug via the Rules Palette, the application should be started in debugging mode. Debugging mode adds a lot of extra information to enable remote debugging, and therefore, generators create extra lines of code.

The PAS.properties file section for settings debugging:

```
#----------------------------------------
# This property allows remote level debugging or not. Yes or No.
#----------------------------------------
```

debug.remoteDebugging=No

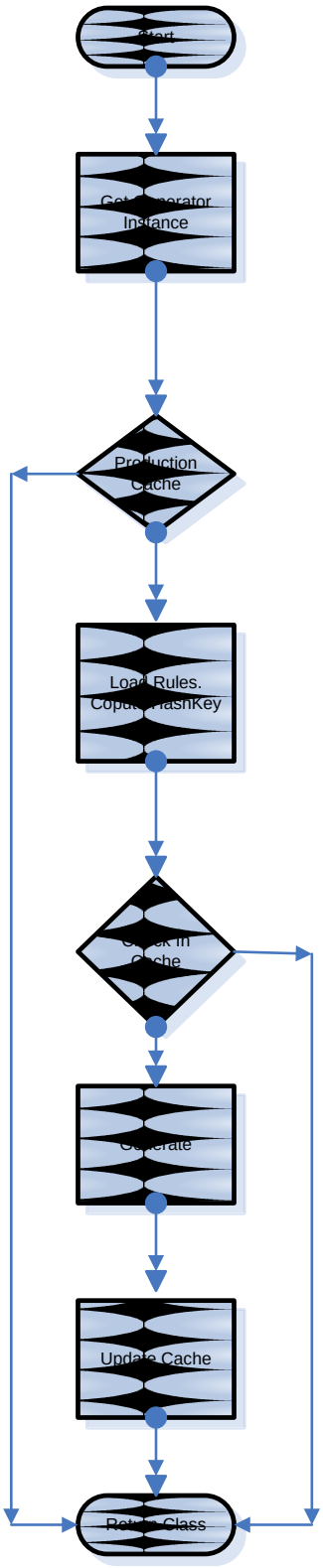
If set to No, then the application will not support remote debugging at runtime. If set to Yes, then remote debugging is supported. It should NEVER be set to Yes in a Production environment.

To support developer debugging of activity processing, Generators can save the generated classes to a local file system if configured in the property file. If debug.IdentiyTranslator is set to Yes, then in the java files generated, at the end of each file, it will add a comment identifying the translator class and the line number that generated that line of code. This is extremely useful in debugging the generated source code and changing it for future needs.

The PAS.properties file section of settings for debugging properties:

```
#----------------------------------------
# Directory to save generated source code.
# This property will be used to debug issues with sre processing.
# Generated source code while processing will be saved in the
# directory specified. Only to be used in Non Production environment.
# debug.identifyTranslator will write comments for every line identifying
# the translator(line number) that generated that part of code.
#----------------------------------------
debug.SaveGeneratedClass=Yes
debug.identifyTranslator=Yes
debug.SaveGeneratedClassDirectory=c:\\temp
```

There are different types of generators for different purposes. A few are described in the next section of this document.



PASTRANSACTIONGENERATOR

PasTransactionGenerator is a Generator specific to OIPA transactions. It understands the rules of the OIPA transaction and generates the classes suited to its processing needs. All generated classes by this Generator extend from PasTransactionBll, which implements the basics of OIPA activity processing.

Logic Flow of the Transaction Generator

Start – Shared rules engine calls the static method in the Generator for activity processing. *PasTransactionGenerator. getTransactionBllForProcessing*

Get Generator Instance - Creates an instance of a generator class per transactionGUID. There is only one instance of the Generator per transaction, but many instances of the Generator for different transactions. This prevents multiple threads calling to process activities of the same transaction type and simultaneously translating the same rules. Only one thread translates the rules for a transaction and other threads, it there are any, wait for the first thread to complete. It then uses the class for processing.

Production Cache – The Generator then looks at the cache to see if a class exists for this transactionGUID. In Development mode, the cache will not contain the key. In Production mode, if the transaction is already translated, it will pick it up and return.

Load Rules, Compute HashKey – If false in the above decision, the Generator loads the rules from the database. It computes the unique hash key for the rules XML.

Check Cache – It then checks in the cache to see if it has a class file for the hash key generated. In Development mode, if the transaction is updated then the hash key will be different and it will force the Generator to translate again.

Generate – It will parse the rules, translate the rules using translators and then compile the generated java classes. It also saves to the file system if specified in the property file.

Update Cache – It updates the cache depending on development or production mode for future use. Future calls with the same transaction GUID and same hash key are not translated.

Return Class – Returns the instance of the generated class to the caller.

SegmentCalculatorGenerator

SegmentCalculatorGenerator is used to create classes at runtime for segment calculation based on the rules. This creates classes specific to the OIPA system. SegmentCalculator follows the same algorithm as the PasTransactionGenerator except that it has only one instance of the generator class for all segment rules where PasTransactionGenerator has one instance per transaction GUID. At any given point of time in a JVM, only one segment calculation can be translated.

Note: If there are multiple requests to retrieve a segment calculator class, one gets through and others are blocked until the class is returned from the cache or translated and compiled.

MathEngineFactory

This is the stand alone math Generator. This class is not named like other classes, which end in the word Generator. It is good to note this, to avoid confusion regarding it being a Generator. The MathEngineFactory loads the rules with only math sections and creates a class that executes the math rules and returns the results. This is an independent math generator that is used by valuation, exposed computation and any module that has math sections that need to execute.

FunctionCallGenerator

FunctionCallGenerator is used to create function code. The generated function rule classes are embedded within the transaction, segment or math classes. These Generators are not thread synchronized because currently they are called from one of the above generators and they are throttled above.

ScreenEventGenerator

There are four types of ScreenEventGenerators: OnLoadGenerator, OnSubmitGenerator, OnClickGenerator and OnChangeGenerators. These are used to process the rules at three different events of the application. They are not related to activity processing but part of the shared rules engine as they involve processing math calculation.

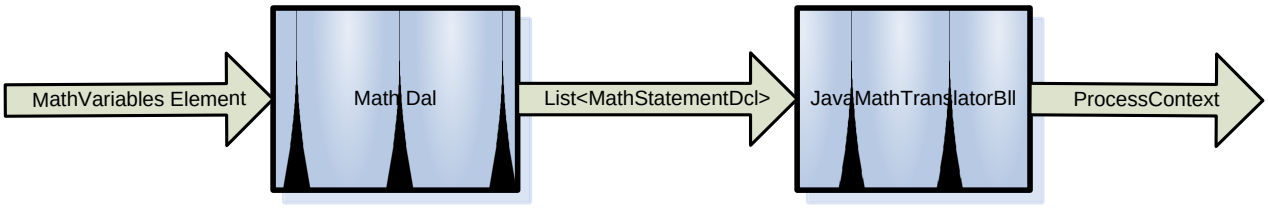
RequirementProcessorGenerator

The RequirementProcessorGenerator generates Java code for requirement processing based on the configuration provided in the Requirement Definition XML. The generator is a part of the shared rules engine but is not actually related to activity processing.

MATH

The Math module is a sub-component of the shared rules engine, which is responsible for executing any math sections in the rules. In a rule, all tags between the `<MathVariables>` element are handled by this sub-component.

Conceptual Math Functionality



Conceptual Math Functionality

1. The Generator that generates java source for the <MathVariables> section calls the MathDal with the location to the MathVariables element in the rules XML file.
2. The MathDal classes parse the element and its sub-elements and create a list `<MathStatementDcl>` and returns it as an output. MathStatementDcl represents the entire tree hierarchy of the math section with loops and MathIF's.
3. The above List `<MathStatementDcl>` is sent to JavaMathTranslatorBll for translation. Each math statement Dcl is translated and the corresponding java code for that statement is set in the MathStatementDcl itself.
4. JavaMathTranslatorBll returns an instance of ProcessContext that contain lots of information. It contains:
    - List of MathVariables declared
    - List of functions called
    - Other structures for dependency and debugging purposes
5. Generators get the ProcessContext and generate the final class with variable declaration, statements and function calls for compilation and execution.

TRANSLATORS

Translator classes are the most important piece of the Math sub-component. Translators are responsible for translating every MathStatementDcl to its corresponding java source code. Translators perform error checking and also code generation for the single XML line.

Each <MathVariable> type that is defined by the TYPE attribute has one or more translators associated with it depending on the operations allowed on the math type and its complexity. MathVariableType.java, an enum, defines the list of all MathVariable TYPE and the corresponding translator classes. JavaMathTranslatorBll iterates through the MathStatementDcl and invokes the corresponding translator with the MathStatementDcl to perform the translation.

Note: Please see the XML Configuration Guide in the OIPA Documentation Library on Oracle's Technology Network for more details regarding XML schemas and definitions used by various OIPA rules.