ORACLE
INSURANCE

Oracle® Insurance Calculation Engine (OICE)

Release Management

Version 9.7.0.0

Documentation Part Number: E51103-01

December, 2013

License Restrictions

Table of Contents

Overview

Purpose

The purpose of this Release Management document is to provide an overview of the Oracle Insurance Calculation Engine (OICE) Release Management functionality. It is intended solely to help current customers understand the capabilities of Release Management, as well as to provide an example of how it might best be used within product releases. It should be noted that each customer may have slightly different needs. As such, it is suggested that customers also work with Oracle Consulting, who may also involve Oracle development resources, to adjust the procedures cited here to best meet individual needs.

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.

Introduction

OICE's Release Management capabilities provide a systematic method of tracking rule changes, as well as a process for packaging and migrating rules from one environment to another. Release Management is an optional tool within the Rules Palette. Using Release Management allows the configuration team to:

- track multiple projects, such as new development initiatives as well as defect/bug remediation and the rules that are affiliated with each
- force the tracking of any rule change into a Configuration Package
- monitor migration status
- bundle packages for a release to ensure that no rules critical to functionality are forgotten or missed during a migration
- facilitate an audit trail and allow the user to view the rules that were moved to environment(s).

At each stage of the rules migration process, tight security privileges are provided to ensure a separation of duties among the configuration team.

Requirements

- Target and source environments must share the same IVS database and it is recommended that they exist on the same track. If the environments do not share an IVS database, then a detached migration should be performed. Even though IVS versions aren't the same, the environments should start at the same OICE baseline. Refer to the Detached Migration section for additional information.
- Turn Release Management functionality on via the Web Application Utility. This is done by a build manager as individual users have no ability to turn release management on and off in the Rules Palette.
- Install the Rules Palette and create security roles for users to control the access to the various stages of Release Management. Sample security roles and the corresponding privileges are outlined in the **Create Security Roles** section.
- Set up the versioning tool. Steps to prepare for versioning are outlined in the **Prepare for Versioning** section.
- Establish the naming conventions that will be used for the Configuration Packages. This is critical to ensure all configurors are well aware of the strategy used when creating Configuration Packages. This helps to prevent rule conflicts within multiple packages. Refer to Step 3a in the **Important Considerations** section on page 16 for additional information on naming conventions.
- Create a generic Configuration Package to hold rules that are deleted from a Configuration Package. Use a descriptive naming convention such as "Orphan Rules". The Configuration Lead will need to review all orphaned rules to ensure they can safely be deleted.

Prepare for Release Management

Create Security Roles

Security plays an important role in the Release Management process. The Security Manager will need to set security levels for configurors and managers who are involved in the Release Management process. Several sample security roles are provided below to demonstrate a typical security breakdown but this can be modified to meet specific business needs. The Release Management folders in the Rules Palette are not visible to users unless they have received a security role with release management privileges.

Sample Security Roles

Configuration Analyst: This role is responsible for creating Configuration Packages. Typical security privileges would include:

- ConfigPackage-Add
- ConfigPackage-ReadyToMigrate
- ConfigPackage-Remove
- ConfigPackage-ViewOwned
- BusinessRules- All Non-Administration Rules-Check-In/Check-Out
- Administration- All Non-Security Administration-Check-In/Check-Out
- Administration- All Non-Security-View

Configuration Lead: This role is responsible for reviewing all Configuration Packages and resolving any rule conflicts. The Orphan Rules Configuration Package must also be reviewed by this role to ensure that any deleted rules are not needed in any other Configuration Package. Typical security privileges would include:

- ConfigPackage-ViewAll
- ConfigPackage-PutOnHold
- ConfigPackage-TakeOffHold
- MigrationSet-ViewAll

Build Manager: This role is responsible for reviewing Migration Sets and creating and building Release Packages. Typical security privileges would include:

- ConfigPackage-ViewAll
- MigrationSet-ViewAll
- ReleasePackage-Build
- ReleasePackage-Create
- ReleasePackage-ViewAll
- Administration- All Non-Security-View

Rules Architect: This role is responsible for managing a high level view of the complete Release Management process and tracking the progress at the various stages. Typical security privileges would include:

- ConfigPackage-ViewAll
- MigrationSet-Create
- MigrationSet-ViewAll
- ReleasePackage-ViewAll
- Administration- All Non-Security-View

Release Manager: This role is responsible for finalizing the Release Package and deploying it to the target environment. Typical security privileges would include:

- MigrationSet-ViewAll
- ReleasePackage-SetReadyForSCM
- ReleasePackage-Deploy
- ReleasePackage-ViewAll
- Administration- All Non-Security-View

Prepare for Versioning

This process should be performed according to the status of the configuration in the affected environments. If active configuration has not yet taken place, then follow the steps outlined in the **Pre-Configuration IVS Initialization Process for Versioning**. If active configuration has already begun in the environments, then follow the steps outline in the **Post-Configuration IVS Initialization Process for Versioning.**

Pre-Configuration IVS Initialization Process for Versioning

This process assumes active configuration has not begun against the affected environments. If configuration has begun and this process was not run beforehand, then please use the **Post-Configuration IVS Initialization Process for Versioning**.

1. Make sure all active environments are set up and properly aligned with your migration strategy in their corresponding Web Application Utility.
2. Stop all Rules Palette configuration and any OICE applications running against the databases where you are planning to implement Versioning and Release Management.
3. Execute the ClearIVS.sql script against the IVS database. These scripts will leave current security intact. The scripts are provided in Appendix A of this document.
4. Choose a baseline OICE database that all other OICE databases will be based upon.
5. Take a full backup of the baseline OICE database.
6. Restore the baseline OICE backup to all remaining OICE databases that will be a part of the IVS/RM ecosystem.

7. Restart all Web Application Utilities.
8. Start the newest version of the Rules Palette and create new environments for each of the OICE applications/databases.
9. Run the Version Generator in "Generate New Versions" mode against the baseline environment.
10. Run the Version Generator in "Generate Reference Versions" mode against all remaining environments (one environment at a time), choosing the baseline environment as the reference.
11. Restart all OICE applications.
12. Login to all OICE application environments to assure that the environment is up and running. Login to all Rules Palette environments to make sure the Rule Palette environments are set up correctly. Verify in all environments that one of each rule type is present and has the correct version associated with it (at this point, all versions will be 1).
13. There is also an optional step to use the **Diff Report** tool in the Rules Palette to confirm that the same information exists in both databases. This tool and the process for using it are documented in the Rules Palette help system within the application.

Post-Configuration IVS Initialization Process for Versioning

This process assumes active configuration has already begun against the affected environments. If configuration has not yet begun, then please use the **Pre-Configuration IVS Initialization Process for Versioning**.

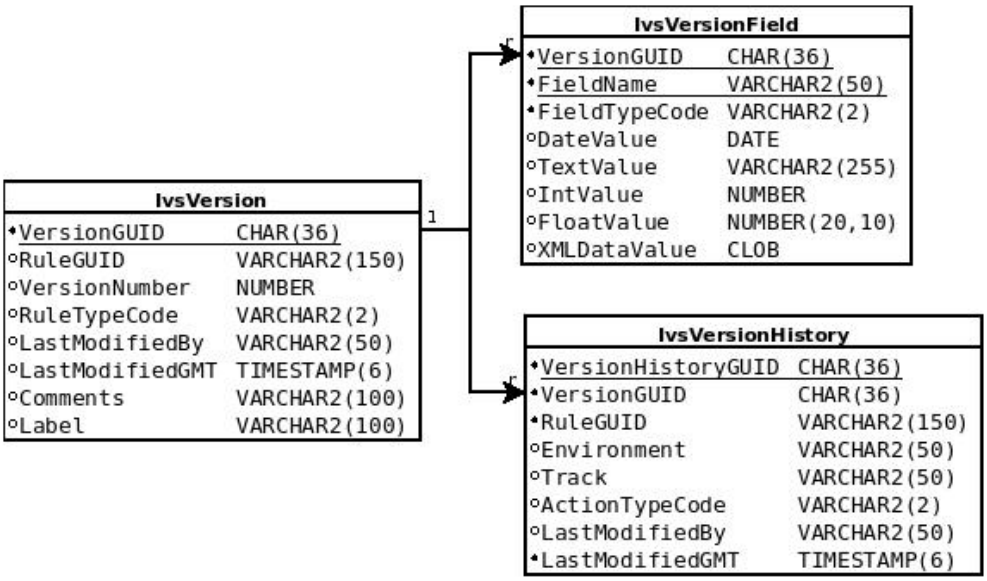
1. Make sure all active environments are set up and properly aligned with your migration strategy in their corresponding Web Application Utility.
2. Notify ALL configurors/testers to stop work on affected environments.
3. Stop all Web Application Utilities and OICE applications running against the databases where you are planning to implement Versioning and Release Management.
4. If you want to retain your current Versioning/Release Management system for audit purposes, take a backup of the IVS database.
5. Execute the ClearIVS.sql script against the IVS database. These scripts will leave current security intact. The scripts are provided in Appendix A of this document.
6. Choose a baseline OICE database that all other OICE databases will be based upon.
7. Take a full backup of the baseline OICE database.
8. Restore the baseline OICE backup to all remaining OICE databases that will be a part of the IVS/RM ecosystem.
9. Restart all Web Application Utilities.

10. Start the newest version of the Rules Palette and create new environments for each of the OICE applications/databases.
11. Run the Version Generator in "Generate New Versions" mode against the baseline environment.
12. Run the Version Generator in "Generate Reference Versions" mode against all remaining environments (one environment at a time), choosing the baseline environment as the reference.
13. Restart all OICE applications.
14. Login to all OICE application environments to assure that the environment is up and running. Login to all Rules Palette environments to make sure the Rules Palette environments are set up correctly. Verify in all environments that one of each rule type is present and has the correct version associated with it (at this point, all versions will be 1).

General Architecture: IVS

Table Structure

Figure 1: IVS Table Structure



How IVS Works

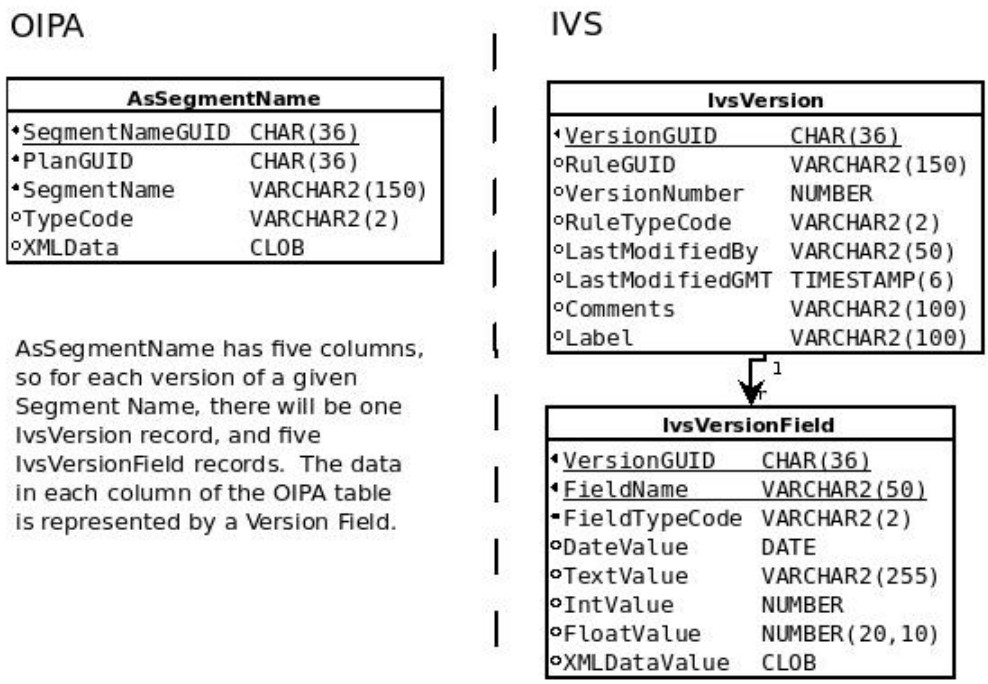
Background

Many items modified through the Rules Palette are tracked by an Internal Versioning System (IVS). This system was designed to track creation, modification and deletion of system rules and configuration. IVS performs the same functions for OICE configuration that Subversion performs for Java code. The reason for choosing to develop our own system instead of interfacing with our clients in-house SCMs was the complexity involved in making such a system work across all the different third party SCMs our clients were using.

Functionality

A version of a rule represents a snapshot of that rule at a given point in time. Each version of a rule consists of data stored in two tables: IvsVersion and IvsVersionField. IvsVersion has a one-to-many relationship to IvsVersionField (see *Figure 1)*. In the standard versioning scheme, each IvsVersionField record represents a column in the OICE table containing the record being versioned. The figure below provides a detailed example of how this works.

Figure 2: OICE Column and IVS Version Field Design



Not all rules are versioned using this design. Non-standard versioning strategies (strategies that do not rely solely on the OICE Column:IVS Version Field design) generally use XML CLOBs stored in Version Fields. The XML Schema is determined by the Strategy team and can vary widely for different types of configuration. Some rules use a combination of the OICE Column:IVS Version Field strategy combined with the XML CLOB strategy. An example of this is Rates. Rates are versioned by AsRateGroup. For each version of a Rate Group, you will see all of the AsRateGroup columns in IvsVersionField, and you will see an extra Version Field that represents the AsRate records within that group. Because there is the potential for an extremely large amount of rates within a Rate Group, the rates are versioned as a delta with the previous version. Only Version 1 of a Rate Group contains all of the rates in the XML Version Field. After that, only differences (deltas) between the rates are displayed.

Version History

IVS is designed to track data across multiple OICE environments. It also keeps a detailed audit trail of all actions performed on each version of a rule. All of this is accomplished through the IvsVersionHistory table.

IvsVersion has a one-to-many relationship to IvsVersionHistory (see *Figure 1*). The following actions in *Figure 3* can be performed on a version of a rule and will be logged in

the IvsVersionHistory table (actions can also be found in IvsCode searching on the CodeName="IvsCodeActionType", Code Value is listed next to the action description in double quotes):

Figure 3 - Version History Actions

Insert "01" - New rule is created	UndoCheckout "05" - Revert modifications/Undo Checkout
Delete "02" - Rule is deleted	Migration "06" - Rule was migrated to specified env/track
Checkout "03" - Rule is checked out	Conversion "07" - Version created by a Conversion Tool
Checkin "04" - Rule is checked in	VersionGenerator "08" - Version created by Version Generator

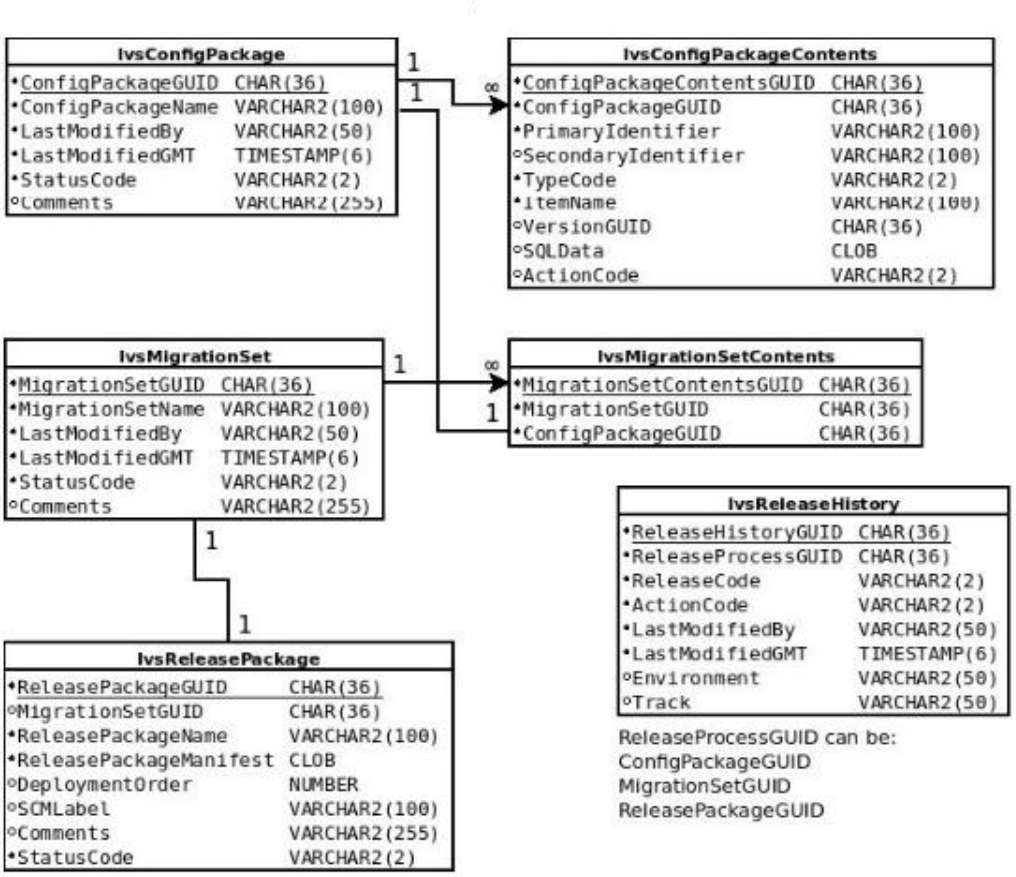
All of these actions are logged on an environmental basis using the IvsVersionHistory.Environment and IvsVersionHistory.Track columns. The environment name should be a simple description of the environment (e.g. "DEV", "TEST", "PROD", etc.). The track is numeric and used to define a flow of data. A good analogy for a track in IVS is a branch in Subversion. Generally, environments labeled as being in Track 1 are part of the development trunk, and all other tracks define separate units of work. With our clients, Tracks 2 and greater are typically used for a new Plan implementation. Environments and tracks uniquely represent any given environment where configuration, testing, or live data may reside. Environment/Track combinations **MUST** be unique for each environment they represent.

Current versions of a rule are tracked in the IvsVersionHistory table. To find the current version of a rule in a given environment, query IvsVersion and IvsVersionHistory with the following criteria: RuleGUID (BusinessRuleGUID, TransactionGUID, etc.), environment, and track. The most recent CheckIn "04" record in the IvsVersionHistory table represents the current version.

General Architecture: Release Management

Table Structure

Figure 4: Release Management Table Structure



How Release Management Works

Background

Release Management is designed as a error proof way of promoting and tracking configuration changes in OICE from the development environment all the way to production.

Release Management tracks changes in a database using **Configuration Packages**, which have a one-to-one relationship with a development issue. When Release Management is enabled, a configuror is required to add all changes to a Configuration Package, which means that all changes are tracked in Release Management. When the time comes to migrate the changes, the available Configuration Packages are displayed in the Rules Palette and added to a **Migration Set** by a Release Manager. The Migration

Set is then added to a **Release Package**, along with other non-configuration components, to be deployed to the target environments.

Definitions of Release Management Functionality

Configuration Packages

Configuration Packages represent a unit of work, or an issue in an issue tracking system (e.g. JIRA). A Configuration Package can contain several rules that were modified as part of an issue. It can also contain non-rule items such as SQL Scripts (DML or DDL), stored procedures, and reports. All Configuration Packages are represented in the database by the IvsConfigPackage table, and the rules that are part of them are represented by IvsConfigPackageContents (see *Figure 4*). This is a one-to-many relationship.

Use of Configuration Packages by configurors is required when Release Management is turned on. A Configuration Package prompt will be displayed in three different scenarios, assuming the rule is not already in an open Configuration Package. If the Configuration Package is not owned by the person trying to access the rule, then a message will appear saying that the rule is locked by an open Configuration Package. The owner's name will be listed in the dialog box and the check-out action will be cancelled. Configuration Package prompt scenarios include **Rule Checkout, Check-in of a New Rule,** and **Rule Deletion**.

Once a configuror has added all modified rules to a Configuration Package and completed work, the Configuration Package is marked as "Ready to Migrate". At this point, the configuror is done with the Configuration Package and will move onto the next issue. When the configuror marks the package **Ready to Migrate**, the system populates the VersionGUID column of each IvsConfigPackageContents record.

In some cases, a configuror may need to release the rules in a Configuration Package due to higher priority issues. The **Put On Hold** feature allows the rules within the package to be released to other configurors. When a rule is released, the changes made to that point remain in the rule. The next configuror can add the needed changes and then migrate his Configuration Package. Now the rule can be re-added to the original Configuration Package and work can resume. It is very important that the rule be unit tested before configuration begins again. This is necessary to ensure the newest rule changes don't conflict with the original configuror's intent for the configuration.

The following type codes related to CPs correspond to the following IvsCode Code Names:

```
IvsConfigPackage.StatusCode: IvsCode.CodeName =
"IvsCodeConfigPackageStatus"
```

A list of statuses available for a Configuration Package appears in *Figure 5* below.

Figure 5 - Configuration Package Statuses

Open "01" - Work is in progress	Migrated "04" - CP has been migrated
ReadyToMigrate "02" - Work is completed	Hold "05" - Work is on hold
AddedForMigration "03" - Added to Migration Set	

The Configuration Lead is responsible for reviewing all Configuration Packages to make sure there are no conflicts. Any rules that were deleted should reside in an Orphan Rules Configuration Package. All these rules should be reviewed to ensure the rules are not needed in any other configuration.

Migration Sets

Migration Sets represent a collection of Configuration Packages that will be associated with a release and migrated from a Development environment to different environmental levels of a company, generally ending in Production. All Migration Sets are represented in the database by the IvsMigrationSet table, and the Configuration Packages that are part of them are represented by IvsMigrationSetContents (see *Figure 4*). This is a one-to-many relationship. The IvsMigrationSetContents:IvsConfigPackage relationship is one-to-one. While IvsMigrationSetContents looks similar to a link table, the relationship cannot be many-to-many because a Configuration Package may only be in one Migration Set.

Because Migration Sets have a one-to-one relationship with a Release Package, a common question is, "Why don't you just add Configuration Packages to the Release Package?" We were looking to separate the duties of organizing configuration into a release. Creating a Migration Set requires in-depth analysis of what changed in the configuration as well as what will be changing in the target environment. This will usually be performed by a Business Analyst or someone with detailed knowledge of the system and configuration. Creating a Release Package requires no knowledge of the configuration and can be performed by an IT department or anyone designated as a Release Manager.

Migration Sets are created in the Rules Palette from the Admin Explorer only when all issues expected to be in the release are complete and their Configuration Packages are in the "Ready to Migrate" status. Migration Set creation is completed using a simple UI that at the present time only has the ability to enter a Migration Set name and select Configuration Packages for inclusion in the release. In the future, we plan to provide a suite of analytical tools for use by the Business Analyst at this step.

The following type codes related to Migration Sets correspond to the following IvsCode Code Names:

```
IvsMigrationSet.StatusCode: IvsCode.CodeName =
"IvsCodeMigrationSetStatus"
```

A list of statuses available for a Migration Sets can be found in *Figure 6* below.

Figure 6 - Migration Set Statuses

Open "01" - MS creation complete	Migrated "03" - MS has been migrated
AddedToPackage "02" - Added to Release Package	

IMPORTANT: Configuration Packages cannot be removed from a Migration Set. SQL Scripts must be used to perform this operation.

Release Packages

Release Packages represent a release of configuration changes, application files, SQL scripts, and other files critical to a release. The Release Package is stored in the database, but the system is also designed to generate an archive file (zip or jar) to represent the release. Currently a Release Package can only be deployed from the IVS database, but functionality to deploy directly from the file is in the planning process.

All Release Packages are represented in the database as the IvsReleasePackage table (see *Figure 4*). There is a one-to-one relationship with the IvsMigrationSet table. The ReleasePackageManifest column stores an XML CLOB that serves as a readable version of the contents of the Release Package. ReleasePackageManifest is for users who don't have access to the database, but want to know what is in the Release Package. It can also be used for the planned "file deployment" functionality described above. There is also a DeploymentOrder column. This column is not yet fully utilized, but the purpose is to check and make sure Release Packages are not deployed out of order. By running a check for un-deployed packages with a Deployment Order that is less than the current Release Package's Deployment Order, we can make sure that the configuration integrity remains intact across all environments.

All Release Packages are created using a Release Package Wizard in the Rules Palette Admin Explorer. Options in the wizard include choosing file type (.zip or .jar), naming the Release Package, adding comments or SCM Labels, attaching executables (WAR files), attaching properties files (SystemInformation.properties), and verifying (but not removing) SQL scripts, stored procedures and report files. The verification steps are simply a confirmation that non-configuration items need to be added to the Release Package. They cannot be removed in this wizard because along the way the Business Analyst who created the Migration Set decided they were required as part of the release.

The following type codes related to Release Packages correspond to the following IvsCode Code Names:

```
IvsReleasePackage.StatusCode: IvsCode.CodeName =
"IvsCodeReleasePackageStatus"
```

A list of statuses available for a Release Package is shown in *Figure 7* below.

Figure 7 - Release Package Statuses

Open "01" - RP was created	Promoted "04" - RP has been promoted
Built "02" - RP archive file was created	Deployed "05" - RP has been deployed
ReadyToPromote "03" - RP is ready for promotion	

Promotion mentioned in statuses 03 and 04 has to do with external Source Control Management (SCM) systems. The idea was to have SCM environments that corresponded to the OICE environments, then to set a Release Package "Ready to Promote", check the Release Package into the SCM environment, "Promote" the Release Package to the target SCM environment, and then deploy the Release Package to the target OICE environment. This process was part of an "external approval procedure," which was an offshoot of Sarbanes Oxley legislation. All of these steps must be performed manually as the Rules Palette currently does not interface with any third party SCMs.

IMPORTANT: Release Packages cannot be rolled back once they are deployed. The Release Manager must manually fix the problem.

Release History

Release History provides an audit trail for all actions that take place in the Release Management process. Release History is represented by the IvsReleaseHistory table (see *Figure 4*). The Release History table shows that action that occured (ActionCode), when it happened (LastModfiedGmt), the user who executed it (LastModifiedBy), where it happened (Environment AND Track), what type of Release Management component the action is being performed against (ReleaseCode), and the identifier for the component (ReleaseProcessGUID).

Every time the status changes on a Configuration Package, Migration Set or Release Package a record is written to IvsReleaseHistory. The same status codes are used from IvsCode that are used for each individual Release Management component.

Detached Migration

When a migration needs to occur between two databases that do not share the same IVS, then a detached migration must be performed. A detached migration begins in much the same way as a typical migration. The configuration packages are created and marked ready to migrate. They are then added to migration sets.

When a release package is created, the first screen has a check box option to create files for detached migration. Clicking this box will allow the release package to migrate outside of the IVS environment of the source environment. After the release package is created, it is built and promoted exactly as it is in a typical migration.

The Rules Palette has a tool that manages the deployment of a detached migration. This tool can only be used in the target environment, after a release package has been built and marked ready to promote. A Detached Migration wizard walks you through the steps to complete the deployment of the release package. When the deployment is initiated, OICE performs conflict validations on each rule prior to writing the rule to the database. If a conflict occurs, you will be presented with a warning message, along with action buttons to support user overrides or a cancel button to abort the detached migration.

Once the deployment is successfully completed, a confirmation message is presented.

Trouble Shooting the Detached Migration

The IVSRELEASEPACKAGE table has a column called DETACHEDFLAG. A value of Y indicates that the package was marked for detached migration.

The location of the detached migration files can be found by referencing the <BuildLocation> tags in the RELEASEPACKAGEMANIFEST column of the IVSRELEASEPACKAGE table.

The ReleaseManifest.xml file contains a list of all items included in the detached migration. Items are grouped by type and are separated by start and end tags that reflect the item type.

Recommended Release Management Process

The Release Management process must be carefully planned and managed. This section contains important considerations that must be addressed when setting up Release Management. A sample Release Management workflow is also included here to provide a tangible example of a working scenario.

Important Considerations

1. All environments must share the same IVS Database. If the same IVS is not shared, then a detached migration must be performed. Make sure the two environments have the same OICE baseline before performing the detached migration.
2. The initial setup of a development track(s) should include setting the environment to the current view of production.
3. Configuration for a given project, which is slated to move to production, will first perform all of its testing between the development environment and its Development QA environment.
    a. Configuration Packages within a Development Track should be named using a consistent naming convention that identifies the purpose of the configuration. A recommendation is that the Configuration Package name include: **Project Name, Configuration Release, Name of the Feature/Function, and the Fix # (if applicable: the first release to QA will not have a Fix #).** An example of a configuration name might be Term2: Build3: PremiumPayment for the first time it is released to QA. Names cannot be reused; they must be unique. If a similar name is needed append a date to the name to make it unique.
    b. For each build or drop to QA within that environment, there will be correlating Release Packages assigned.
    c. At the end of final sign-off by QA for a given build or drop that is set to go to production, there will be multiple Release Packages.
4. When configuration is ready to move from a Development Track to production, a production pipeline Release Package should be created. This will include all of the configuration changes that are to move to production.
    a. The Production Pipeline Release Package will be made up of the rules within the Configuration Packages of the Release Packages to the QA environment within the Development Track.
    b. The Release Packages to the QA environment within the Development Track should be examined in reverse order from which they were migrated to QA.
    c. For each Release Package, the rules within each of the Configuration Packages should be added to a new, single Configuration Package for

the Production Pipeline. In this way, it will be clear when a rule has been migrated multiple times, and only the current version of the rule will be included.

d. Once all of the QA Release Packages have been examined, and all of the rules within the Configuration Packages have been added to the new Production Pipeline Configuration Package, the Production Pipeline Release Package may be created.
e. All Plan and Company data being added or updated will require scripts, and should be added as “Non-Palette” release management items.
f. All updates to Chart of Accounts, Requirements, Data Dictionary and Translation Editor require table moves and should be added as “Non-Palette” Release Management items.

5. When migrating to the Production Pipeline Development environment, it will be necessary to merge all of the rules and check for conflicts. This is not a systematic process – the reason being the Production Pipeline represents the current version of production. Changes to the actual production environment may have occurred since the start of the Development project that is now ready to be migrated to production. Diff Report functionality provided in the Rules Palette can be used to compare the development track with the production track. Use the summary diff report option to view all differences. The Rules Palette help has comprehensive instructions for using this feature. Configurors must review to determine if the rules being migrated are to be:
    a. Added: New rule that is simply added to production.
    b. Deleted: Rule that did exist that is to be removed from use.
    c. Edited: Changes made to an existing rule.
6. When the merge of Development rules into the Production Pipeline occurs, a new single Configuration Package should be created within the Production Pipeline environment. As the rules are being analyzed and merged into this environment:
    a. An Add of a rule will require the creation of an INSERT script that will then be used to insert the rule into the Production Pipeline Development environment. Once the rule has been inserted, the configuror can then go to the rule and check it out within the Production Pipeline Development environment. At check-out, the configuror will add the rule to the Production Pipeline Configuration Package (This will now be noted as an UPDATE within the Configuration Package.) An additional step is required to then modify the contents of the Configuration Package for that rule to change the “UPDATE” to an “INSERT.” This requires a script to be run in the database.

Script Instructions:
First, find the ConfigPackageContentsGUID by picking it from the list of contents:

```
SELECT * FROM IvsConfigPackageContents WHERE
ConfigPackageGUID IN ( SELECT ConfigPackageGUID FROM
IvsConfigPackage WHERE ConfigPackageName = '{Insert
Config Package Name Here}' )
```

Then insert it into this query (be careful to use the **ConfigPackageContentsGUID**, **NOT** the ConfigPackageGUID):

```
UPDATE IvsConfigPackageContents SET ActionCode = '02'
WHERE ConfigPackageContentsGUID = '{Insert
ConfigPackageContentsGUID Here}'
```

To see the different actions available for a Configuration Package Contents record, you may run the following query:

```
SELECT * FROM IvsCode WHERE CodeName =
'IvsCodeDBTransactionType'
```

b. A delete of a rule requires some research. Analysis will determine if the rule ever existed in Production. If it did not, then there is no additional work to be done, and the rule is NOT added to the Production Pipeline Configuration Package. If the rule DID exist in production and it is determined that it should be deleted in the Production Pipeline, then the rule will need to be deleted within the Production Pipeline Development track and added to the Configuration Package.

Analysis Instructions:
Determining if a rule exists in the Production Pipeline requires a Find Best Match query. You cannot query by GUID because if a rule was created in the Production Pipeline AND the Development Track while both were in separate development cycles, the same rule will have different GUIDs. To perform the Find Best Match query, you'll need to identify the unique index of the table. For example, the AsTransaction table's unique index consists of PlanGUID and TransactionName, so the find best match query would be:

```
SELECT * FROM AsTransaction WHERE PlanGUID = '{Insert
PlanGUID Here}' AND TransactionName = '{Insert
TransactionName Here}'
```

If this returns a record, you must delete that record and all of its references (e.g. if you're deleting a transaction, make sure to delete any

Business Rules with that TransactionGUID attached to them, along with all other related records).

c. An edit, or merge, of the Production Pipeline and the QA rule will require the rule to be checked out in the Production Pipeline, added to the new Production Pipeline Configuration Package, merged with work completed, and checked back in within the Production Pipeline environment. This does not require any extra analysis, but be careful not to overwrite any existing functionality while adding your new functionality.
d. All Plan and Company data being added or updated will require scripts, and should be added as “Non-Palette” Release Management items.
e. All updates to Chart of Accounts, Requirements, Data Dictionary and Translation Editor require table moves and should be added as “Non-Palette” Release Management items.

7. Because of the potential, inherent volatility associated with a merge, full regression testing of your Production Pipeline is recommended before the final deployment to the Production environment.
8. When all of the rules have been merged into the Production Pipeline development environment, and there is a single Configuration Package containing all rules that are to be migrated through the Production Pipeline, follow normal Release Management deployment procedures, and there will be no more merge work required.
9. After the final release to the Production environment, the Development track used for the Production environment should be reset to match Production, setting it up for further project work.

Sample Release Management Workflow

The following section provides a practical example of Release Management workflow.

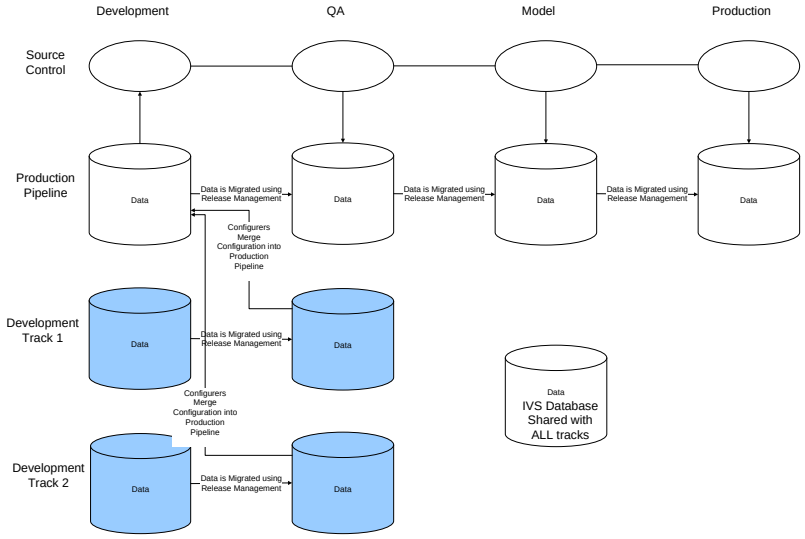
Assumptions

- All rules databases share the same IVS, including production.
- After the development environments, the clients can follow any configuration/path that they choose. Clients should work with Oracle Consulting to help determine the best setup to meet their needs.

1. Create Environments:
    a. Development: Where configuration work on a project is completed. Unit testing is done in this environment.
    b. QA: Where full, non integration, QA (testing) is completed. No development work is done in this environment.
    c. Model: Where full QA (testing) is completed, including integration. This environment mimics exactly what the production environment will look like after project deployment. No development work is done in this environment.
    d. Production: Environment that holds all production data, and where customer servicing is performed. No development work is done in this environment.

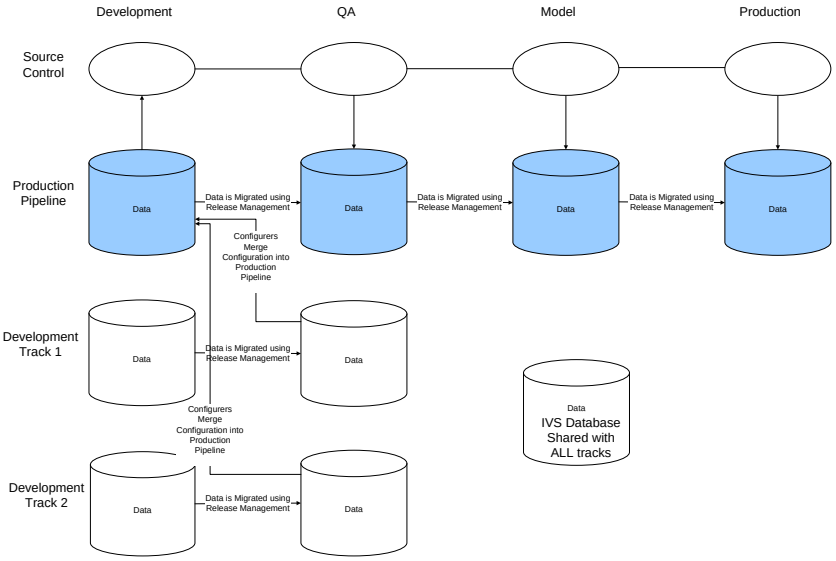
2. Begin Testing and Configuration in Development Tracks: Development Track 1 and 2 are single environments where project work that will eventually migrate to production will take place. Within each development track is a QA environment. The configuration to a single QA environment will come only from its development track. When fully tested the QA is migrated to the Production Pipeline.

Figure 8: OICE Release Management – Multi-Track Migration Process



3. Promote Tested Configuration to the Production Pipeline: The Production Pipeline is a track that is used to fully test the developed configuration in a setting that mimics production. The Production Pipeline environment represents the current state of the production environment.

Figure 9: OICE Release Management Production Pipeline



a. The development piece of the Production Pipeline will have all the configuration of Production, but will not have all the integrations. It will be updated with configuration from the various tracks (represented as Development Track 1 and 2). Configuration is merged into this track to ensure that needed configuration is not overwritten, as may be the case due to prior releases to production. Unit testing will be done in the Development environment.

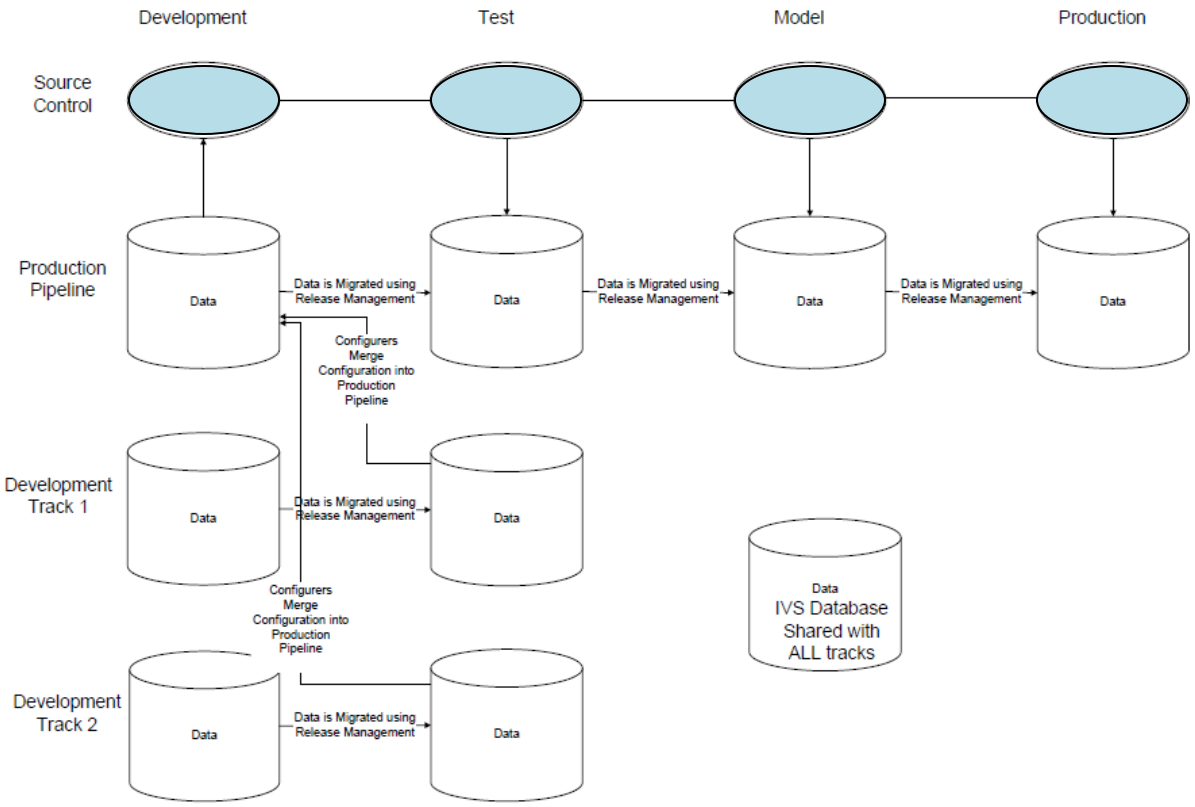
b. The QA environment in the Production Pipeline will contain the full testing of all configuration, which will mimic the new Production configuration.

c. Along with the general QA environment, the Production Pipeline also includes a Model environment, or an environment that mimics the production environment, including all integrations.

d. After all testing has been completed, the configuration will move to Production. At that point in time, all environments within the Production Pipeline track will contain exact copies, not including the integration connections.

4. The Source Control line in the diagram below represents the potential for the migrations to be integrated with an external source control system if desired.

Figure 10 – OICE Release Management – Source Control



Appendix A: ClearIVS SQL Script

The script below should be executed against the IVS database when preparing for versioning. It will leave current security intact.

DB2 Database

```
delete from ivsconfigpackagecontents
go
delete from ivsconfigpackage
go
delete from ivsmigrationsetcontents
go
delete from ivsmigrationset
go
delete from ivsreleasepackage
go
delete from ivsreleasehistory
go
delete from ivsversion
go
delete from ivsversionfield
go
delete from ivsversionhistory
go
SELECT
(SELECT COUNT(*) FROM IvsConfigPackageContents) AS CPC,
(SELECT COUNT(*) FROM IvsConfigPackage) AS CP,
(SELECT COUNT(*) FROM IvsMigrationSetContents) AS MSC,
(SELECT COUNT(*) FROM IvsMigrationSet) AS MS,
(SELECT COUNT(*) FROM IvsReleasePackage) AS RP,
(SELECT COUNT(*) FROM IvsReleaseHistory) AS RH,
(SELECT COUNT(*) FROM IvsVersion) AS V,
(SELECT COUNT(*) FROM IvsVersionField) AS VF,
(SELECT COUNT(*) FROM IvsVersionHistory) AS VH
FROM sysibm.sysdummy1
go
```

Oracle Database

```
delete from ivsconfigpackagecontents
go
delete from ivsconfigpackage
go
delete from ivsmigrationsetcontents
go
delete from ivsmigrationset
go
delete from ivsreleasepackage
go
delete from ivsreleasehistory
go
delete from ivsversion
go
delete from ivsversionfield
go
delete from ivsversionhistory
go
SELECT
(SELECT COUNT(*) FROM IvsConfigPackageContents) AS CPC,
(SELECT COUNT(*) FROM IvsConfigPackage) AS CP,
(SELECT COUNT(*) FROM IvsMigrationSetContents) AS MSC,
(SELECT COUNT(*) FROM IvsMigrationSet) AS MS,
```

```
(SELECT COUNT(*) FROM IvsReleasePackage) AS RP,
(SELECT COUNT(*) FROM IvsReleaseHistory) AS RH,
(SELECT COUNT(*) FROM IvsVersion) AS V,
(SELECT COUNT(*) FROM IvsVersionField) AS VF,
(SELECT COUNT(*) FROM IvsVersionHistory) AS VH
FROM DUAL
go
```

Appendix B: Future Enhancements

The following section contains an explanation of the future enhancements planned for Release Management.

Cross Track Merge Support

Problem: Current practices only allow for configuration migration in a linear fashion through a single development track. To move code between Development tracks, the configuration must be manually merged (in the Rules Palette, check out configuration and merge changes, and check back in). This is a time consuming and error-prone process.

Proposed Enhancement: The solution is to provide a suite of analytical tools to assist with the merge. These tools include a full environment difference report, an individual rule comparison tool and potentially the ability to visually merge rules within the individual comparison (although this may be a rather large effort).

Migration Rollback

Problem: Currently, once a Release Package is deployed, there is no way to "roll back" that deployment in the event of a critical error created by a configuration change.

Proposed Enhancement: Create an action for rolling back entire deployments.

Migration of Non-Rule Items

Problem: Support for adding non-rule items such as SQL scripts, application WAR files and properties files exists in Release Management; however, there is not a lot of strategy surrounding this functionality. We currently support adding the files to the system, but have no strategy for potential deployment of them. This part of the system may also require further testing as it has not been used by anyone yet.

Proposed Enhancement: Develop a strategy for the ability on deployment to execute SQL scripts, copy properties files to the application server, and potentially deploy the application WAR file. Test the current capabilities of the system in regards to these components.

Release Management History Viewer

Problem: The Release Management system provides a very detailed tracking system. This system is only accessible by running SQL statements through a query analyzer application.

Proposed Enhancement: Create a Release Management History Viewer application in the Rules Palette. This would allow users to view the Release Management history, across all environments, and filter it based on multiple criteria.

Clarify Release Management Component Statuses

Problem: Statuses on different components in the Release Management system are currently global statuses. Statuses of individual components are currently not tracked in each environment. For example, in the current system, once a Release Package is deployed, it is marked as "Deployed", and stays in that status for the rest of its life, even though it has not been deployed to all the environments.

Proposed Enhancement: Actions performed against these components can be tracked using the Release History table (rather than the individual status codes on each component's table). Per environment statuses using the history should be reflected in the Rules Palette.

More Robust Migration Set Creation Feature Set

Problem: Migration Sets were created as a point where a Configuration Lead or someone with complex knowledge of the configuration could review and filter the data that was part of a migration. This way, someone without knowledge of the system could create the Release Package and run the deployment(/s). Currently, only the most basic Migration Set creation function exists.

Proposed Enhancement: Implement functionality during Migration Set creation for Configuration Package analysis, duplicate rule checks, conflict identification, cross-environment rule comparisons and referential integrity analysis.