Oracle® Retail Active Retail Intelligence

Operations Guide
Release 13.2.4.0.1
E35693-01

June 2012

ORACLE

Oracle® Retail Active Retail Intelligence Operations Guide, Release 13.2.4.0.1

Primary Author: Nathan Young

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

The following restrictions and provisions only apply to the programs referred to in this section and licensed to you. You acknowledge that the programs may contain third party software (VAR applications) licensed to Oracle. Depending upon your product and its version number, the VAR applications may include:

(i) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.

(ii) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Mobile Store Inventory Management.

(iii) the software component known as **Access Via™** licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.

(iv) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

The VAR Applications contain trade secrets of Oracle and Oracle's licensors and Customer shall not attempt, cause, or permit the alteration, decompilation, reverse engineering, disassembly or other reduction of the VAR Applications to a human perceivable form. Oracle reserves the right to replace, with functional equivalent software, any of the VAR Applications in future releases of the applicable program.

.

Contents

Send Us Your Comments

Oracle Retail Active Retail Intelligence Operations Guide, Release 13.2.4.0.1

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document.

Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

> **Note:** Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Applications Release Online Documentation CD available on My Oracle Support and www.oracle.com. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

Oracle Retail Operations Guides are designed so that you can view and understand the application's behind-the-scenes processing. Including such information as the following:

- Key system administration configuration settings
- Technical architecture
- Functional integration dataflow across the enterprise
- Batch processing

Audience

Anyone who has an interest in better understanding the inner workings of the Active Retail Intelligence system can find valuable information in this guide. There are three audiences in general for whom this guide is written:

- System analysts and system operation personnel who need information about Active Retail Intelligence processes.
- Integrators and implementers who are responsible for implementing Active Retail Intelligence.
- Business analysts who need information about Active Retail Intelligence processes and interfaces.

Related Documents

For more information, see the following documents in the Oracle Retail Document Template Release 13.2.4.0.1 documentation set:

- *Oracle Retail Active Retail Intelligence Release Notes*
- *Oracle Retail Merchandising Implementation Guide*

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 13.2) or a later patch release (for example, 13.2.1). If you are installing the base release and additional patch and bundled hot fix releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch and bundled hot fix releases can contain critical information related to the base release, as well as information about code changes since the base release.

Oracle Retail Documentation on the Oracle Technology Network

Documentation is packaged with each Oracle Retail product release. Oracle Retail product documentation is also available on the following Web site:
http://www.oracle.com/technology/documentation/oracle_retail.html

(Data Model documents are not available through Oracle Technology Network. These documents are packaged with released code, or you can obtain them through My Oracle Support.)

Documentation should be available on this Web site within a month after a product release.

Conventions

Navigate: This is a navigate statement. It tells you how to get to the start of the procedure and ends with a screen shot of the starting point and the statement "the Window Name window opens."

```
This is a code sample
    It is used to display examples of code
```

1

Introduction

This guide is designed to explain the administrative processes that support the operation of Oracle Retail Active Retail Intelligence (ARI).

How and When to Use this Guide

None of the processes described here should be undertaken until after ARI installation is complete. Appropriate administration of ARI is critical to its successful operation, and although many of the tasks, once in a production, are primarily the responsibility of the database administrator, significant collaboration between the database administrator and business analyst is required throughout the rule definition process. Database administrators should read and understand this guide after installation and before doing any additional ARI work. Business analysts should read and understand this guide before creating any ARI rules.

One of the first things to do after reading and understanding this guide is to make sure the ARI Options table values are all set correctly (see Appendix C for details of the options).

2

Process Overview

As discussed in the Overview sections of the online help, ARI is only a tool that helps with several steps of a larger process, which, for context, is reviewed here. This larger process, to implement a new business process, or existing process modification, involves first gathering requirements and conducting user walkthroughs, designing possible implementation strategies, reviewing impact, choosing a strategy and conducting additional walkthroughs. This is followed by development of rules, test implementations and user acceptance. Finally, the rules are moved into production. The sub-process relevant to ARI administration is essentially the same in development, test and production.

More than a single, specific process, the sub-process relevant to database administrators is a set of processes that can be linked in a number of different ways depending on the operational environment. Factors such as, whether the database is shut down regularly or not, and how the shutdown occurs (shutdown immediate or otherwise), will impact exactly how these sub-processes should be implemented.

ARI Shutdown

ARI uses the DBA_JOBS queue to control the background processes essential to exception detection and validation. ARI also uses generated code in these detection processes. (A more detailed architectural overview can be found in Appendix A.) Before making *any* DDL changes in any ARI schema or in any schema monitored by ARI, including recompiling packages, alter tables, creating new functions, or even running the ARI code generator, it is critical that the Exception Validation Engine (EVE) be halted from processing exception candidates. (Stopping EVE can be done using one of the many stop methods described along with the other APIs in Appendix B.)

EVE uses persistent variables to control multi-threading and other aspects of its operation. Because of this, EVE should be stopped before bringing down the database and restarted when the database is brought back up. If EVE is not stopped, either due to an unexpected shutdown or shutdown before EVE is cleared from the DBA_JOBS queue (EVE may take a few minutes to finish processing after the shutdown script is run), the shut down request should be sent immediately after the database is restarted. Once it has cleared the DBA_JOBS queue, EVE can then be restarted.

Metadata Modification

As a rule, metadata should be synchronized with the actual DDL and other objects (Oracle Forms, for example) it describes at all times. Because such instantaneous synchronization is essentially impossible, the next best thing is that these tasks are performed in immediate sequence. This is not terribly difficult since the DDL and Forms should change very little during production, and when they do, non-administrative users usually must be logged out while the DDL changes are taking place.

> **Note:** While the DDL and metadata changes are made, EVE should NOT be running.

In series with actual DDL changes and metadata modification, the code generator must also be run immediately while database users are still logged out. DDL changes can invalidate generated code, but the generated code can only be corrected accordingly after the metadata is changed, so this third thing, code generation, must be done in sequence with its precedents. EVE should not be running during code generation either, as will be discussed.

A possible exception to this rigorously limited circumstance under which metadata can be changed is when new actions or functions are added to the system. By the time you have gone through development and test, you should have the expectation that migration of the new packages or Forms will be routine. Create the metadata for these objects, before shutting down EVE, and complete the other two steps in the process:

1. Actually install the changes
2. Run the code generator.

If you choose to make modifications to the metadata, other than the times when EVE is shut down, it is critical that you only add or change the metadata that you just added and actually add the new code prior to running the metadata generator.

Rule Construction and Modification

Rules can be constructed or modified at any time except while the code generator is running. New rules appear in the system only after the code generator is run. Modifications involving an exception or event end date require that the code generator be run before they will take effect. Other modifications, such as the linking between exceptions and event, or schedule to either exceptions or events are dynamic, and will take effect as soon as they are applied. There is no significant administrative task here, but this process is highly dependent on the code generation process that it is worth highlighting. Rule construction drives the need for code generation, so database administrators and business analysts may like to stay coordinated about how and when the new rules are likely to be created.

Import Export Tool (IET)

IET is a tool for copying rules into and out of an ARI instance. It allows ARI consultants to supply retailers with pre-packaged rules. It allows retailers, to replicate rules between different testing and production instances. This is preferable to manually recreating rules within each ARI instance. Rules are exported to .xml documents in a database-independent form. From these documents, the rules can be imported to a different ARI instance, where IET will attempt to resolve metadata references within the rule to corresponding metadata in the new ARI instance. ARI consultants can also package up supporting actions and data that will be added to new ARI instance with the rule during import. Select Enter to create blank line under the last line of text. This is where you will insert the new graphic.

Code Generation

Two processes require code generation. Rule construction/modification, and metadata changes. After either of these occurs, code generation is required. Code generation is required immediately after metadata changes (except in the instance of new metadata where delayed code generation may be deferred), and in a reasonable timeframe after rule construction/modification. (For details on executing code generation, see the API details in Appendix B.)

One of the issues with when to run code generation is how soon it needs to be run after rule construction or modification. The code generator builds supporting code for exceptions and events for all new exceptions and events configured to start before some future time; specifically, now (the time the code generator is started) plus the number of hours defined by the FORWARD_ GENERATION_HOURS option. (For forward generation details reference Appendix C on configuring ARI options).

Obviously code cannot be generated constantly, so forward generation is done in anticipation of a business analyst defining an exception or event several hours, or even a few days, before they actually want it to take effect. Generally, significant planning and design effort are put into defining exceptions and events, so planning well enough ahead that, the code generator need not be run immediately is typical in a production environment. Certainly the code generator can be run at any time that EVE is stopped, but ideally in a production environment it is run at an off-hours time so that any issues can be handled with minimal impact on the production environment.

Exceptions and events can be linked even after code generation, both linked to each other and to schedules. However, rules cannot be modified after code generation, and the consequence of forward generation is that changes to effective dates do not take effect until the code generator is run again. An exception or event set up to start immediately will only start after the code generator is run. Once scheduled to stop in three weeks, but you want to stop it now instead, can be stopped as soon as you set the end date andrun the code generator again.

Both of these cases can be handled as they occur by simply stopping EVE and running the generator on demand, but a typical configuration might be to forward generate by 30 hours and run the generator daily at approximately the same time (as a scheduled job even) every day. This typical configuration does not preclude on-demand generation as well (alternatives and other issues are in Appendix C).

ARI Start

Once the first rules are built and the code generator run, EVE and the other continuously running processes (Scheduler) should be active whenever the database is running. Running the appropriate ARI start program can start all of these programs. If only EVE is shut down (perhaps during a code generation run) and in need of starting, then a different program can be used to start EVE only (see Appendix B).

Summary

The following diagram summarizes four possible options (vertically) for performing key routine ARI tasks. The critical processes for which EVE is not running and users are logged out are highlighted in darker boxes, specifically Change Metadata and Modify DDL. The left-hand column shows the status of the DBA_JOBS queue each step of the way.

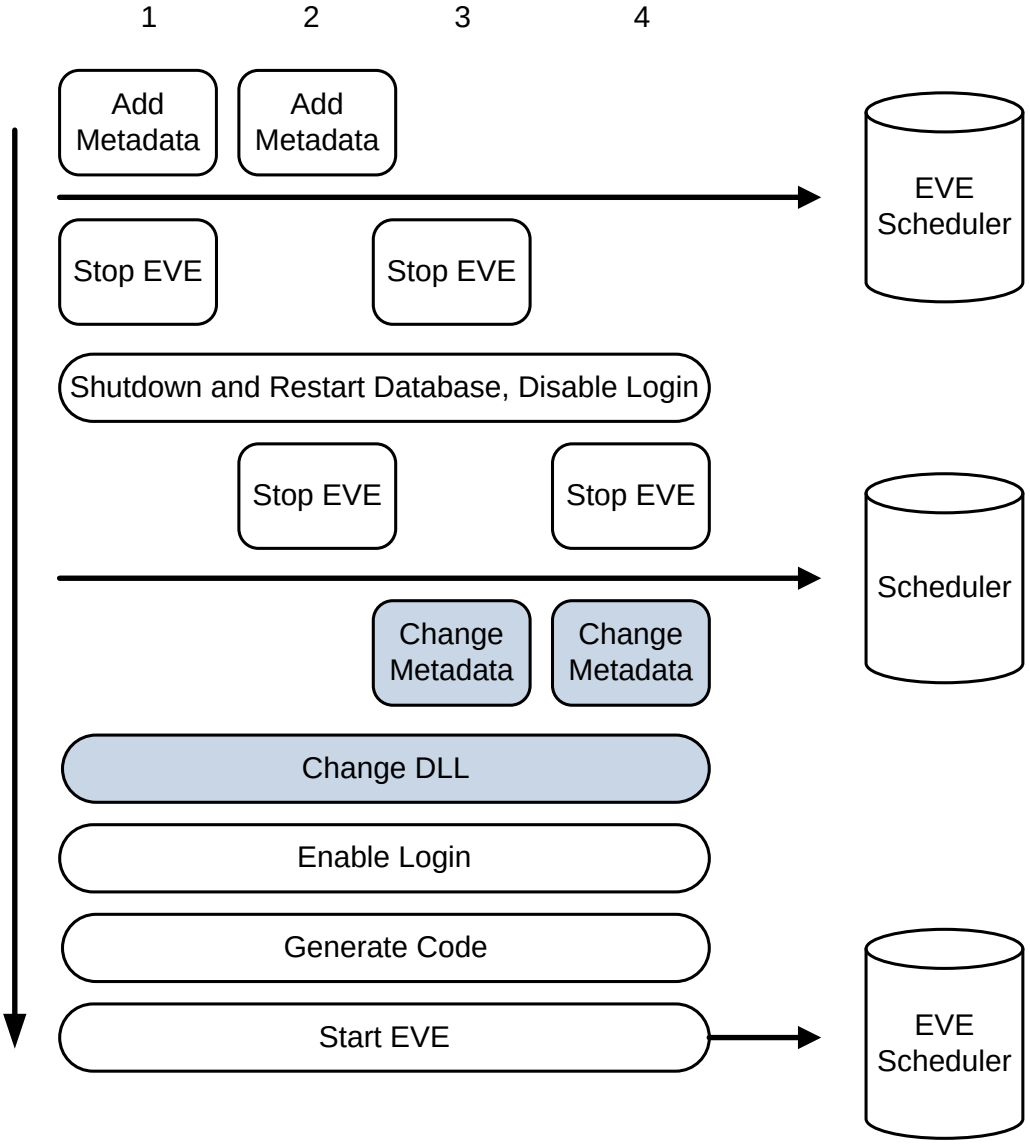


Diagram: DBA_JOBS Queue

3

Process Details

ARI has several continuously running processes whose progress should be checked periodically. The processes write to an activity log and an error log table, ARI_ACTIVITY_LOG and ARI_ERROR_LOG respectively. The continuously running processes and the additional processes they spawn are queued in the DBA_JOBS queue. Both business analysts and database administrators will need to be familiar with these tables and processes, and at least the database administrator, with some escalation procedure to the business analyst, will want to review the tables and processes on at least a daily basis.

ARI Logs

Understanding Log Types

ARI_ERROR_LOG should usually be empty. When it is not, then the errors should be reviewed. Certain types of errors may be non-critical, but a repeated number of such errors may indicate some action is required. Everything written to the error log is also in ARI_ACTIVITY_LOG (the activity log), but the activity log contains many more details of what the system is doing and can be helpful for troubleshooting.

Setting the Log Level

ARI backend processes have message levels assigned to log entries. Level 1 messages are the most important, with level 3 messages being at a fine detail level. Typically we only run level 2 at Oracle Retail, but level 1 may be sufficient for production. The log level is cached for a given database session, so to change it requires starting a new session. In the case of EVE or the Scheduler this means stopping them, resetting the LOG_LEVEL option on ARI_OPTIONS (see Appendix C) and creating a new database session when restarting them.

Reviewing the Log

When the error log is empty the activity log usually does not need review (unless things are not happening as the way it should be, in which case lack of recent entries in either log may indicate ARI is not running). When any errors have been debugged the logs can be truncated.

Purging the Log

There is no automated process for purging the error and activity logs; however, these tables can become quite large if they are not truncated periodically. This truncation is at the discretion of the business analyst and database administrator reviewing the logs. In production daily truncation after review, it should be a standard procedure.

DBA_JOBS Queue

DBA_JOBS will usually contain a scheduler job, an EVE job and a number of validation threads (spawned by EVE) and a number of batch scans and event reevaluation jobs (spawned by the scheduler). Broken jobs indicate something not working correctly. Broken validation threads typically mean that EVE is not going to be spawning as many threads as it could because it will be waiting for the broken thread to return. After troubleshooting, broken jobs should be removed (DBMS_JOB.REMOVE) and, if any of the jobs were validation threads, EVE should be shutdown and restarted.

Scheduler

The scheduler lives in the DBA_JOBS queue. Every five minutes it determines what ARI schedules are due for execution, and generates batch exception scans and event reevaluation blocks, and inserts them into the DBA_JOBS queue. If an exception has multiple schedules and is already being scanned by a function still in queue, that exception will not be added to the new scan if it is scheduled again. The scheduler should remain in the DBA_JOBS whenever the database is running after ARI is installed. It is not harmful for the schedule to be out of the queue, but scheduled jobs will get backed up until it is restarted and will not run on schedule.

Multiple instances of this job should not be running. You must decide from a scheduling standpoint, whether to remove it at shutdown (and again at startup if the typical shutdown is a shutdown immediate) or whether just to leave it in the queue past shutdown (it should start up when the database restarts). In the latter case the DBA_JOBS should be checked daily to make sure it is still an active job (this can be validated by an entry in ARI_ACTIVITY_LOG approximately every five minutes indicating the scheduler is running looking for things to do.

Exception Validation Engine (EVE)

EVE uses the REALM_QUEUE_CTL table to track which realms it is currently processing and which ones to do next. EVE creates validation threads that work on individual realms. These threads get added to DBA_JOBS and when they are complete they tell EVE they are finished so that even can updated the control table and create a new thread on a different realm. Throughout all of this, EVE and its validation threads write to the log extensively.

EVE and its validation threads should be removed from the DBA_JOBS queue before shutdown, or, if they are not because the shutdown is immediate, they should be removed on restart. This is necessary because EVE uses memory persistent information to control its threading process. The issue is primarily one of conserving system resources. Other undesirable behaviors can also result from having too many additional validation threads that EVE creates when not properly stopped either before shutdown or immediately after restart. Before shutting down or running the code generator, it is important to check the DBA_JOBS queue to make sure that EVE is actually done; since its shutdown procedure may not be immediate.

Code Generation

Please note that for code generation, EVE should not be running and the metadata and DDL and Forms must all be synchronized when run. Provided these conditions are met it can be run anytime, though it is generally recommended that it be scheduled periodically and during off-hours to reduce the impact of any code generation errors that might occur. The code generator logs heavily. For troubleshooting it is sometimes helpful to stop both the scheduler during code generation just to simplify reading the log activity.

History Purge

The history purge process should be scheduled often, perhaps even daily. Unless it produces an error it should not require any special attention. History purge removes events older than the history retention days specified in the event definition.

Administrator Groups

ARI uses special user groups, an error group and an analyst group. These groups must always have at least one user each. Both the key database administrator and business analyst should be added to each group before any rules are put into production. If not these users, then someone who will be involved in the daily administration of ARI *must be added before building any rules.* The error group is the assignment target group for all events that enter an error state; the analyst group is the assignment target group for all events that are assigned to otherwise empty groups.

Both groups may also be useful for additional analyst and error monitors that you might like to create with ARI (or that Oracle Retail might in some future release deliver or help build from standard templates during a consulting engagement). The analyst group might be used to build a security function for a custom modification to enforce application security on access to the ARI administrative forms.

> **Note:** Such access level modifications are allowed in spite of the general ceveat that all custom modifications of ARI are unsupported – such a modification is not strictly a functional modification.

4

Product Integration

ARI is a tool intended for use with one or more applications, so integration with other products is an inevitable post-installation configuration requirement for useful production operation. Generic integration issues include creating metadata for the systems ARI will work with, linking ARI into the main application presentation interface of these systems (or not), and multi-language support. Some specific assistance is provided to simplify integration with Oracle Retail's Merchandise System (RMS).

Metadata

Once the seed data has been installed, metadata needs to be created for the systems that ARI will work with. Metadata is the foundation of all ARI rule building, so it is critical that it be accurate. Metadata maintenance is a database administrator task that should become part of the routine that goes with changing Form specifications or table definitions. Fortunately, this does not occur very often in a production system, so it should not be a particularly time consuming task.

Metadata maintenance must be done in sync with DDL changes. Users should not be in the system trying to work with ARI Alerts during the gap in time between when DDL changes are made and when the system is updated. Furthermore, EVE cannot be running during this time. The exception to this rule is when metadata is being added to the system either initially or even after production use of the system has begun. In this case, the metadata can be added during production hours provided the code generator is not run until after the DDL changes are made. This means for initial setup of metadata the entire process can be done during production (and you do not have to log users out for 3 days while you do it).

> **Note:** It is not necessary that the entire system be described in metadata, but the metadata that is described needs to be accurate.

This can be a time saver since you only need to create metadata for the rules you plan to build. There may be some tables in the system that you will require to select data from or monitor, and hence there will be more Forms and PL/SQL actions that you will not need. Analysis before implementing an ARI rule should clearly identify what physical data entities and actions will be needed. So creating appropriate metadata, if it was not already created for another ARI rule, could be treated as part of the rule development process.

Oracle Retail Merchandise System (RMS)

For owners of the Oracle Retail Merchandise System, Oracle Retail has pre-defined the metadata for all of the tables in the system. This metadata can be populated (provided it is the first metadata added after running the seed data scripts that are required in the ARI Installation Guide) by executing in order the scripts provided: mts_parm_type.sql, mts_realm.sql and mts_parm.sql.

Multilanguage Support

The technical infrastructure of ARI supports languages other than English. The software can efficiently handle multiple languages. Tables have been added to ARI to accommodate internationalization. The retailer sets up the user's language preferences on the ARI_USER_ATTRIB table. At login, the LANG column on the ARI_USER_ATTRIB table determines the user's language setting and displays the code string associated with it. ARI has a fail/safe mechanism built into the code. If the user's preferred language is not found, then ARI displays English in the user interface.

Note: A retailer has the two options below regarding internationalization when installing the application. See the *ARI Installation Guide* for the procedures related to each.

- English and multiple secondary languages
- Install English first and then update with a translated language (fully translated non-English installation). No secondary languages are installed when your primary language is one other than English.

Translation

Translation is the process of interpreting and adapting text from one language into another. Although the code itself is not translated, components of the application that are translated include the following:

- Graphical user interface (GUI)
- Error messages

The following components are not translated:

- Documentation (online help, release notes, installation guide, user guide, operations guide)
- Batch programs and messages
- Log files
- Configuration tools
- Reports
- Demonstration data
- Training materials

The user interface for ARI has been translated into:

- Chinese (Simplified)
- Chinese (Traditional)
- Croatian
- Dutch
- English
- French
- German
- Greek
- Hungarian
- Italian

- Japanese
- Korean
- Polish
- Portuguese (Brazilian)
- Russian
- Spanish
- Swedish
- Turkish

Key ARI Tables Related to Internationalization

Several tables were created to handle displayable text that can also be translated.

If the retailer creates a new form, a new menu, or a new object on a form, then the retailer will need to populate these tables with the corresponding information. If the retailer customizes the information in any of the tables ARI_FORM_ELEMENTS, ARI_FORM_ELEMENTS_LANGS, ARI_MENU_ELEMENTS, or ARI_MENU_ELEMENTS_LANGS, the base_ind field in customized records must contain 'N'. Any record with BASE_IND=N will be preserved in a temp table during future patches.

ARI_FORM_ELEMENTS

This table is used for screen display and holds the master list of items for all forms whose labels/prompts are translated. This information will always be in English. The BASE_IND=Y means that the item is part of the base Oracle Retail code set. BASE_IND =N indicates that the item was added as part of retailer customization. Anything with the BASE_IND =N will be preserved at upgrade time on the FORM_ELEMENTS_TEMP, but the retailer is responsible for moving the data back to FORM_ELEMENTS.

ARI_FORM_ELEMENTS_LANGS

This table is used for screen display. This table holds translated values for labels/prompts on forms. This information is in a language that is defined on the lang column of the ARI_USER_ATTRIB table. All users see data from this table; the retailer may customize the text of a given field. The access key for a button is defined by filling in the DEFAULT_ACCESS_KEY field. At runtime, that character will be marked in the string, and function as the access key. Any time the retailer changes the DEFAULT_LABEL_PROMPT or DEFAULT_ACCESS_KEY, the BASE_IND should be updated to N because it is not part of the base language translations provided by Oracle Retail. Anything with the BASE_IND=N will be preserved at upgrade time on the FORM_ELEMENTS_LANGS_TEMP, but the retailer is responsible for moving the data back to FORM_ELEMENTS_LANGS.

ARI_MENU_ELEMENTS

This table is used for screen display. This table holds the master list for all menus whose items are translated. This information will always be in English. The access key for a menu option is defined by using the ampersand (&) before the character that is the access key in the default description. The BASE_IND=Y means that the item is part of the base Oracle Retail code set. BASE_IND=N indicates that the item was added as part of retailer customization. Anything with the BASE_IND=N will be preserved at upgrade time on the MENU_ELEMENTS_TEMP, but the retailer is responsible for moving the data back to MENU_ELEMENTS.

ARI_MENU_ELEMENTS_LANGS

This table is used for screen display. This table holds the values for all menus whose items are translated. This information will be in a language that is defined on the lang table. Even English language users see data from this table, as the retailer may customize the text of a given menu option. Any time the retailer changes the LANG_LABEL, the BASE_IND should be updated to N because it is not part of the base language translations provided by Oracle Retail. Anything with the BASE_IND=N will be preserved at upgrade time on the MENU_ELEMENTS_LANGS_TEMP, but the retailer is responsible for moving the data back to MENU_ELEMENTS_LANGS.

ARI_FORM_MENU_LINK

This table is used for screen display. This table holds the intersection of form and menu files, mapping each form to the menu that it displays.

Presentation Interface

A key feature of ARI is to be able to be notified in your operational applications that an alert has occurred in ARI. To that end ARI provides an API set that will enable you to put a button on the toolbar of yours Forms applications. The button can represent whether any new (undeterred) alerts exist for the user logged into the database, and pressing it will launch the ARI Alert Viewer. For non-Forms applications, it is possible to bypass the Forms library and simply access the PL/SQL procedure that does all of the work. (For details on the API, consult Appendix B.)

ARI has a startup form (aristart.fmb) that allows access to the ARI forms. This form is primarily intended for administrative, full functionality access users to get into ARI. The integrated end-user entry point into the system will typically be through a notification button on the toolbar of the applications that ARI is monitoring, as just described. However, if that entry point is not used, you likely will want to add role-based security to the menus in the ARISTART form to restrict end-user access.

> **Note:** The ARI launch/notification button already exists in RMS, and the API interface is already installed.

To make the interface operational, simply drop the PL/SQL portion of the interface (ARI_INTERFACE_SQL) from the RMS product schema and replace it with a synonym to that same package in the ARI schema.

Integrating ARI with Oracle Retail Workspace

For information about Oracle Single Sign-On and Oracle Retail Active Retail Intelligence (ARI) 13.2.4, see the latest *Oracle Retail Active Retail Intelligence Installation Guide*.

For information on how to integrate ARI with Oracle Retail Workspace, see the latest *Oracle Retail Workspace Implementation Guide*.

Oracle Retail Workspace is a supported configuration of Oracle WebCenter Spaces 11.1.1.5 for Oracle Retail. For the Oracle Retail 13.2.x enterprise, Oracle Retail Workspace replaces previous versions of Oracle Retail Workspace. There is no more packaged Oracle Retail Workspace code, only configuration instructions for Oracle WebCenter Spaces 11.1.1.5.

The Oracle Retail Workspace configuration utilizes the external application functionality and the application navigator taskflow of the Oracle WebCenter Framework to configure ARI in Oracle WebCenter Spaces.

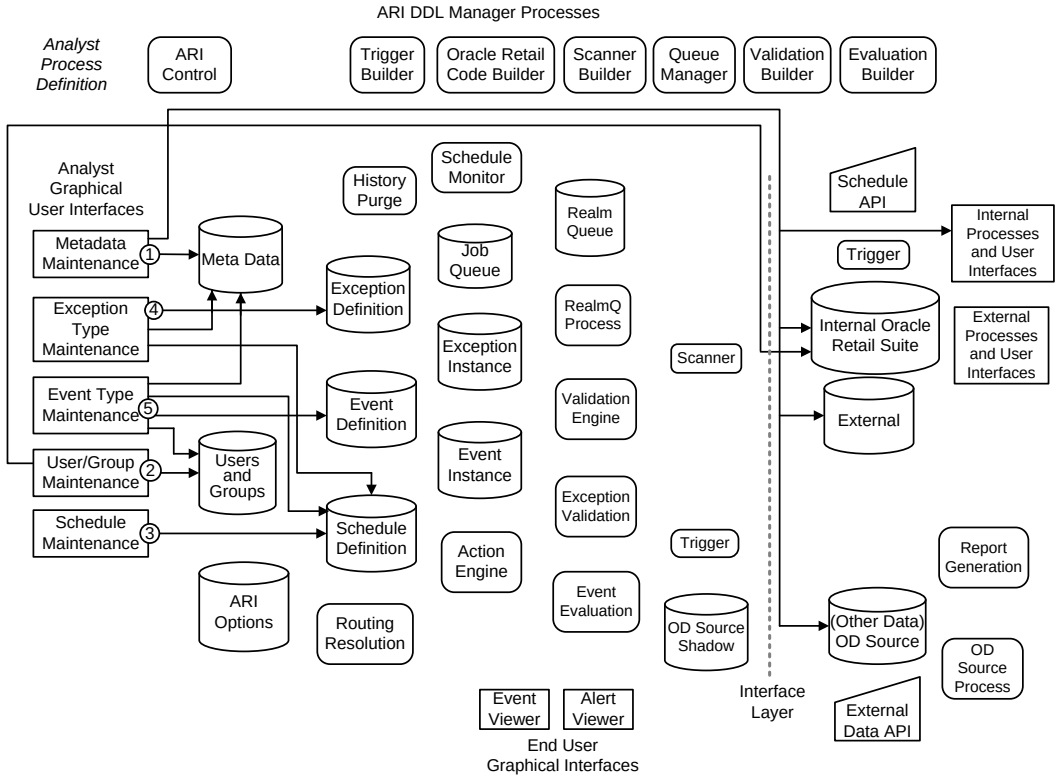
A

Appendix: Architectural Reference

This section provides diagrams and descriptions of the processes that drive ARI. These processes are grouped according to the main functional uses of ARI.

Analyst Process Definition

Before ARI will perform any functions, several processes must be completed to tell ARI what to do. The following diagram and description walk through these processes.



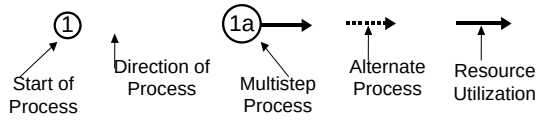
① Set Up and Maintain Metadata

② Set Up and Maintain Users and Groups

③ Set Up and Maintain Schedules

④ Define Exception Types

⑤ Define Event Types



Process Diagram: Analyst Process Defintion

Descriptions of the processes shown in the diagram follow. The numbers in the headings correspond to numbered processes in the diagram.

1. Metadata Maintenance

Database administrators keep metadata and Oracle DDL synchronized. In a production environment, the DDL should seldom, if ever, change. So, once set up, maintaining the metadata is not a time-consuming task. Metadata for new functions/actions to support ARI processes defined by business analysts can be set up and maintained by either a DBA or a business analyst.

2. User/Group Maintenance

Performed by business analysts, user and group maintenance supports event assignment and event supervision assignment. Users entered here must be defined as Oracle users in the database.

3. Schedule Maintenance

Schedules indicate when a task will occur. These are maintained by business analysts and assigned to exception and event definitions to determine when periodic exception scans and automatic event re-evaluations will occur.

4. Exception Type Maintenance

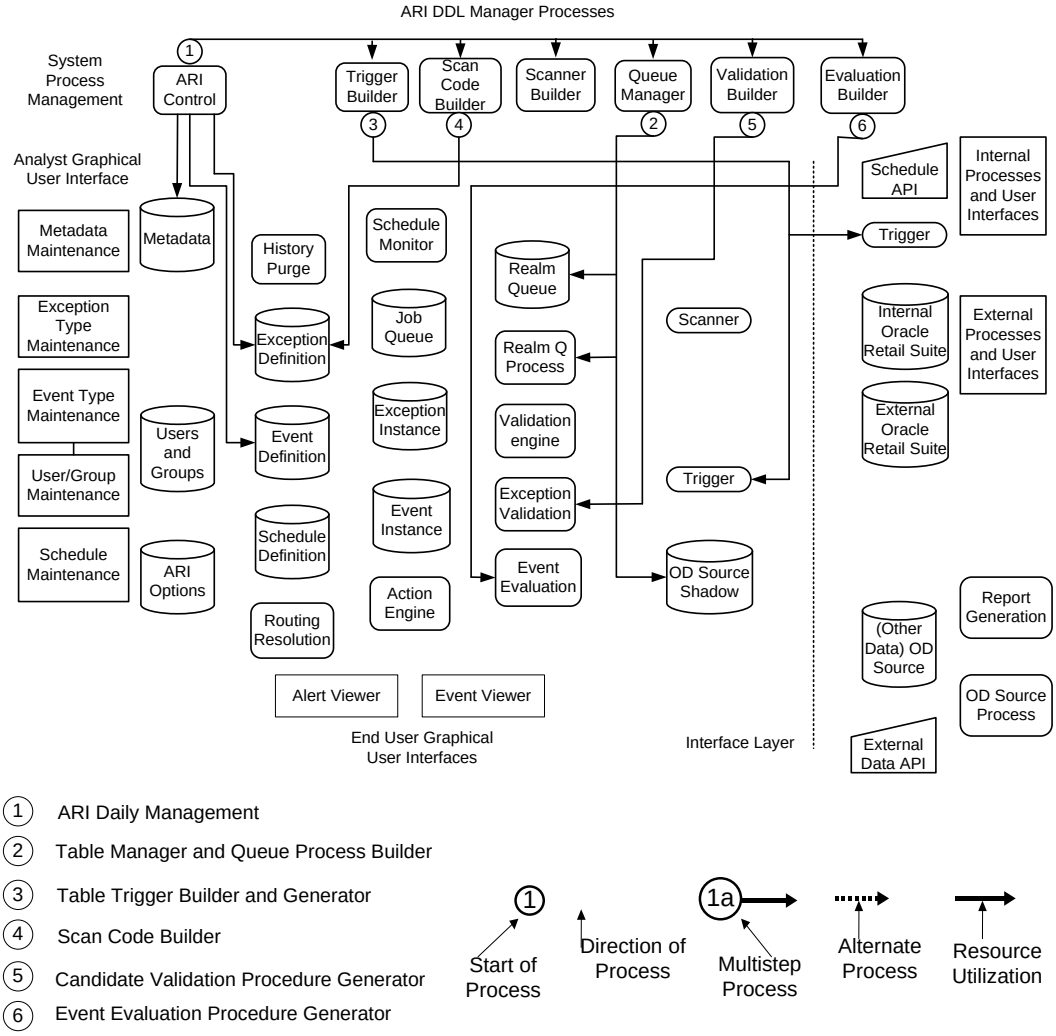
Business analysts maintain exception definitions. They define when to monitor for specific data conditions, what the minimal actionable data conditions are, and what event(s) should be created to help resolve the exception.

5. Event Type Maintenance

Also performed by business analysts, event definition maintenance determines the presentation of exception-related information in the form of alerts. An event contains an alert definition and an exception link, plus information about which users are notified, what actions are available to resolve the event, and what new data conditions will determine that the event is, in fact, resolved.

System Process Management

Using analysts' definitions, ARI prepares to monitor exceptions and present events.



Process Diagram: System Process Management

Descriptions of the processes shown in the diagram follow. The numbers in the headings correspond to numbered processes in the diagram.

1. ARI Control

The daily ARI control program is run at the beginning of the batch window. After all users are cleared, it shuts down the currently active ARI processes, rebuilds exception and event management code (processes 2-6), and then restarts the ARI processes.

2. Queue Manager

This process creates a realm queue table for every monitored realm (if one does not already exist) and deletes those realm queue tables no longer needed. It also builds/drops (as appropriate) realm queue procedures, one per queue, to manage queued data. Finally, it builds the shadow tables that hold data input from external (non-Oracle) data systems.

3. Validation Builder

Drops and creates triggers to monitor real-time exceptions on the Oracle Retail suite that is part of the same DDL as ARI. Also drops/creates triggers on the external source shadow tables that make externally sources data not real-time monitored but trickle-monitored. (The data is moved into the queue real-time as it is received).

4. Scan Code Builder

The scan code builder prepares appropriate clauses to select only rows of interest (based on the exception definition) from tables that are monitored via periodic scanning. This data is stored with the exception definition for easy retrieval by the Scanner builder.

5. Validation Builder

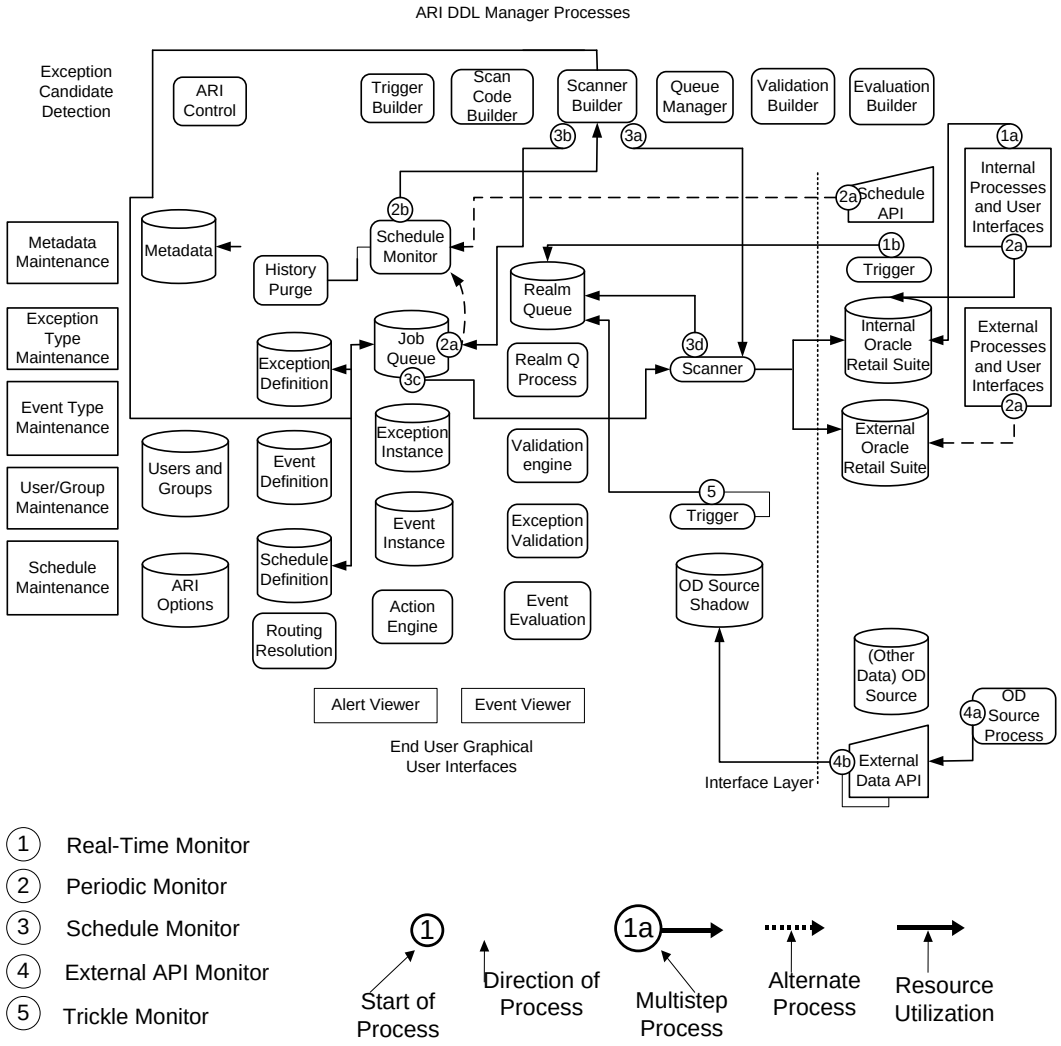
Exception candidates must be validated to determine whether an exception exists. This involves selecting additional information from other parameters and processing the conditions on those parameters. The procedure generator creates this business logic in a static package.

6. Evaluation Builder

Once an exception candidate is known to exist, it can create one or more events. Underlying data conditions may change the way an event should be presented (what actions are available) or who should be dealing with it. The reevaluation logic is encapsulated in a static package.

Exception Candidate Detection

To minimize performance impact on other systems, ARI first identifies candidate exceptions by imposing some of the exception conditions on the data set being monitored. This is done in order to filter out those that could be exceptions, based on complete condition evaluation after additional information is fetched, from those that could not be data states of interest, irrespective of what other information might be gathered up during the exception process. Candidates are queued and processed as systems resources are available.



Process Diagram: Exception Candidate Detection

Descriptions of the processes shown in the diagram follow. The numbers in the headings correspond to numbered processes in the diagram.

1. Real-Time Monitor

When a user makes changes to the database, triggers write candidates directly into their corresponding realm queue.

2. Periodic Monitor

When changes occur in the database that cannot be monitored in real-time due to high volume of changes type considerations or just because the change is time-based (that is, the data is aging), periodic monitoring must be used. The actual monitoring occurs either when a signal is sent to the schedule monitor from the schedule API or when the schedule monitor is periodically prompted by the Job Queue, finds a schedule in need of execution, and runs all of the monitors associated with that schedule.

3. Scanner Builder

The scanner builder assembles the scan code for all exceptions scheduled to occur simultaneously, and places a job in the Job Queue to run those scans. As threads become available for processing, the Job Queue initiates the scans that populate data representing exception candidates into the realm queue tables.

4. External API Monitor

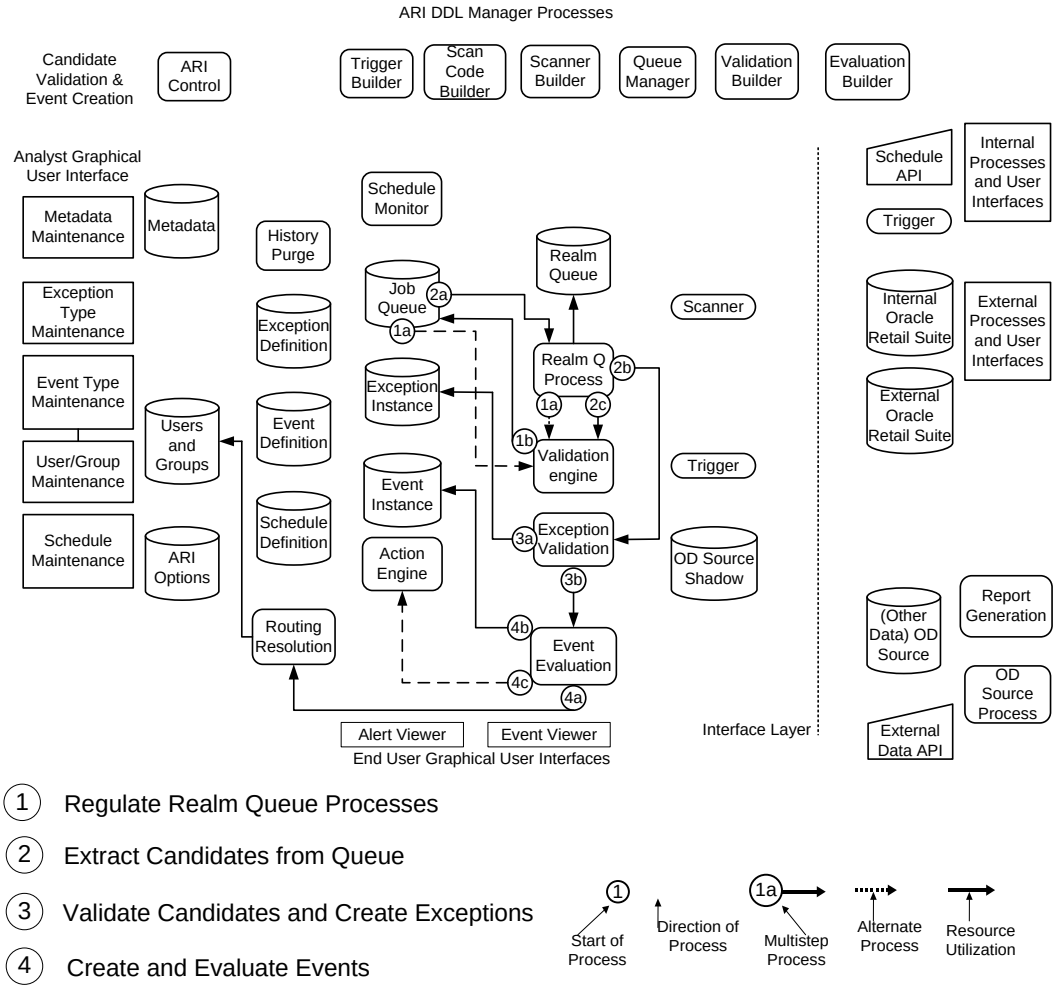
This API is used to feed in data from an external (non-Oracle Retail) system. As the API is called from the external system it writes data to an appropriate shadow table within ARI.

5. Trickle Data Monitor

The trickle data monitor takes data as it arrives from non-Oracle systems through an appropriate monitor and pre-filters it, putting the candidates into the appropriate realm queue.

Candidate Validation and Event Creation

Once candidates have been filtered, it is necessary to validate the exceptions and create an event with which the valid ones are associated.



1. Regulate Realm Queue Processes
2. Extract Candidates from Queue
3. Validate Candidates and Create Exceptions
4. Create and Evaluate Events

Process Diagram: Candidate Validation and Event Creation

Descriptions of the processes shown in the diagram follow. The numbers in the headings correspond to numbered processes in the diagram.

1. Validate Engine

The validation engine regulates realm queue processes. The Job Queue occasionally attempts to start the validation engine because; it is self-terminating when there are no candidates to process. Once started, the validation engine places realm queue processing jobs into the queue (rotating through the queues and checking whether any candidates exist). The validation engine will only queue up a finite number of jobs at any given time, as set in the ARI Option setting "EVE_NUM_THREADS". Once a realm queue job finishes (either because it runs out of candidates or exceeds the time allocated by the validation engine for processing jobs in the queue), it notifies the validation engine, which can then queue up more jobs.

2. Realm Queue Process

The Job Queue starts these processes when threads become available. They read candidate exceptions and call the generated validation procedures for each candidate. After each candidate, the time-slice allocation is checked. When that allocation is exceeded, realm queue processing is terminated and the validation engine is notified.

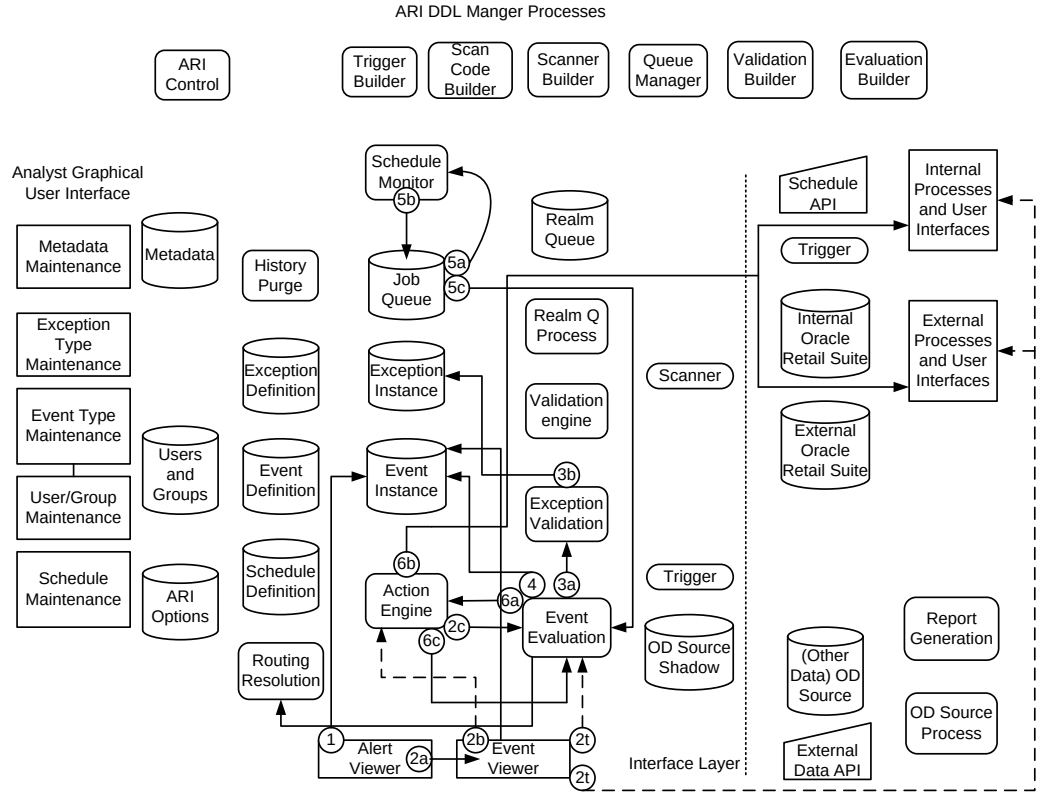
3. Exception Creation and Validation

Valid candidates become exception instances, and then event evaluation is invoked to determine which events should be created or, if they already exist, re-evaluated.

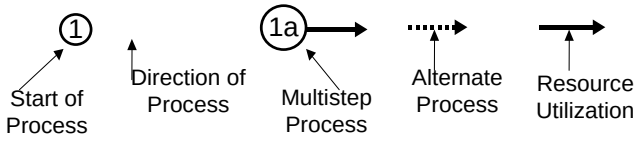
4. Event Creation and Evaluation

Event instances for each event type linked to the exception instances type, must be either refreshed, if an event with the same key values already exists, or created. Evaluation occurs to determine alert routing. Then, the instance is created and, if the appropriate rules are met for a particular instance, an action may be taken automatically.

User-Initiated and Automated Event Resolution



1. Alert and Filter Management
2. Event Management
3. Exception Reevaluation
4. Event Reevaluation
5. Schedule Driven Reevaluation
6. Action Process



Process Diagram: User Initiated and Automated Event Resolution

Descriptions of the processes shown in the diagram follow. The numbers in the headings correspond to numbered processes in the diagram.

1. Alert Viewer

End users initially view events as alerts, which are just event summaries, in the Alert Viewer. This viewer allows sorting and categorization by event type, state, priority, occurrence data, and even whether the user has deferred (marked as having been viewed) the alert. In the Alert Viewer, events are grouped by type state and priority with a count of how many of each type of state and priority exist.

2. Event Management

The Event Viewer shows all events of a single type and state at any given time, and also shows all the details of those events. The user initially is shown the most recently refreshed event data (which may or may not be very recent, depending on whether it is done periodically as defined in the event type). The user may choose to reevaluate the events before proceeding, since re-evaluation may show that some events are already resolved. The user may also drill into the monitored systems to view even more information than what is presented on the event. In addition, the user may take an action to resolve the event or move it along in its process. If an action is taken, the event is still reevaluated beforehand to ensure that the event is still in the state in which the action is valid.

3. Exception Reevaluation

Because the data source for many event parameters is an exception, before an event is reevaluated, its associated exception instances are revalidated. This process is just like validation, except that the exception itself is treated like a new candidate, and is removed from the event and deleted if it is found to be invalid.

4. Event Reevaluation

Once the exceptions are revalidated, event reevaluation refreshes the event parameters and processes its defining rules to determine whether the event is resolved, and should be closed, whether it is in the same state as it was in, and to whom it should be assigned and with what priority. If the reevaluation is user-requested either directly or as a consequence of the user attempting to take an action, and if the state and user assignment are unchanged, then the action can be executed, or if user requested, the user may continue to view and act on the event. Because the Event Viewer shows only a single state at a time, events that change state, although the same user may still own them, will not continue to be seen in the view of the event's previous state.

5. Schedule Reevaluation

Evaluation can be user initiated via one of the previously described processes and may also occur on a scheduled basis. As with scheduled scans for exception candidates, the Job Queue notifies the scheduler to look for tasks that need to be done, which in turn places the tasks in to the Job Queue. As threads become available, these evaluation jobs, specific to an event type, are started. Each job loops through all open instances of their event type and initiates the reevaluation process.

6. Action Execution

Actions are always executed after an evaluation occurs. If an action is user-requested, the event is always evaluated first. Alternatively, as a result of an evaluation, whether prompted by a user request or a schedule, an action may be taken automatically if the event is so defined. After the action is taken, the event is always reevaluated, which may prompt another action, and so on. A well-defined event should never get into an infinite loop, but it is theoretically possible. Therefore, checks are in place to prevent an infinite evaluation-action-evaluation loop from occurring.

B

Appendix: API List

Error and Activity Logging

There are PL/SQL APIs related to error and activity logging that can be incorporated into scripts or used in other contexts.

Changing the Log Level

Within a particular database session, calling ari_error_logging.set_log_cutoff_level can change the log level. The set value will override the ARI options setting within that session unless the function is called again. This can be a useful debugging tool.

Stopping and Starting the Backend

The ARI Backend includes these master processes: the Scheduler and EVE. These processes live in the DBA_JOBS queue from which they run periodically. The Scheduler also creates Batch Scans that it adds to DBA_JOBS, and EVE creates Validator Threads. EVE manages the Validator Threads, and the Batch Scans manage themselves (by simply disappearing from the queue when they are done executing). There are several start and stop methods for the master processes that may be added to database start and stop scripts or executed on a regular schedule, as detailed in the Process Overview.

Starting the Master Processes

ari_control_sql, start_ari calls the start methods for EVE and the Scheduler. It must be run as the ARI master schema owner to ensure that all of the processes will have appropriate privileges when they execute. This is typically placed in a start up script, or, if you prefer to let the Scheduler simply live in the queue at all times, it may only be executed very rarely with the second start method (Start EVE) being the one that is scheduled.

Starting EVE Only

ari_control_sql.start_eve calls the start method for EVE. It must be run as the ARI master schema owner to ensure that EVE will have appropriate privileges when it runs.

Stopping the Master Processes

ari_control_sql.stop_ari calls the stop methods for EVE and the Scheduler. It must be run as the ARI master schema user to ensure that the jobs are actually removed from DBA_JOBS. This is typically placed in the database shutdown script, though often daily bouncing just uses a shutdown immediate, so it is placed in the database start script instead to ensure appropriate shutdown before EVE and the other master processes are restarted.

> **Note:** The Validator Threads must have completed and exited the queue (which can have a slight delay after the Stop Process is run) before EVE is restarted.

When EVE is terminated it signals the Validator Threads to abort and exit the queue, but it can (rarely) take the threads a few minutes to acknowledge the signal. This delay may be either that, (a) they are queued but not yet running, in which case they must first start running as other threads are killed and become available, or (b) they are in the process of opening a cursor.

Also note that Batch Scans are not removed by this script, which is generally what is desired since Batch Scans removed from the DBA_JOBS queue mean that those exceptions contained in the scan will not be scanned again until their next scheduled date. Batch Scans can and should survive database outages. Individual scans can be removed using DBMS_JOB.REMOVE if their execution is meant to be halted (the corresponding database server process will have to be killed as well).

Finally note that if all you typically do is start and stop EVE, leaving the other master processes running, then you will want to schedule the second stop method (that stops EVE only).

Stopping EVE Only

ari_control_sql.stop_eve calls the stop method for EVE. It must be run as the ARI master schema owner to ensure that it will remove the process from DBA_JOBS correctly.

Stopping All ARI Processes

ari_control_sql.stop_all_ari is the same as stopping the master processes, but also kills all of the Batch Scans in the queue. In production this is almost never used, because you do not want to lose those scans, this is more of a development utility. During development an undesirable scan or set of scans may be created by mistake, in which case, killing all processes may be the simplest thing to do, rather than removing them one at a time. Often this is done in conjunction with a quick database shutdown and restart to save the trouble of finding the appropriate database server processes to kill, which may be necessary since jobs in the queue once started run even if they are removed from the queue.

Code Generation

Along with EVE, this is the most critical process to ARI. The code generator must be run as the ARI master schema user to ensure that all of the code can be created properly. Also, it must only be run when EVE and its associated Validator Threads have all been removed from the DBA_JOBS queue. Other operational issues associated with code generation are discussed throughout the Process Overview, such as scheduling it on a periodic basis or running it on demand. Re-summarizing, code generation is tied up very closely with DDL changes and EVE. Whereas users cannot be logged in during DDL changes, they can usually be logged in during code generation. The exception in the above case is when DDL changes in the metadata impact any database objects used by any exceptions or events. In such a case, before the user logins are enabled, the code generator must be run again, after the metadata changes are made.

Scheduler

The scheduler checks to see which scheduler are currently due to execute and creates Batch Scans, and anonymous Event Revalidation Blocks to be added to the DBA_JOBS queue. The scheduler is typically started and stopped via one of the many ARI start/stop control processes, but those methods can be accessed stand-alone for custom scripting etc. The scheduler also contains an API for sending signals to set the next execute date for signal-driven schedule to the current date and time, thus causing the schedule to be executed the next time the scheduler checks schedules due for execution (approximately once every five minutes).

Starting the Scheduler

ari_auto_scheduler.start_scheduler adds the check_schedule procedure (the Scheduler) to the DBA_JOBS queue. This procedure must be run as the ARI master user to ensure that check_schedule has the appropriate privileges when it executes. Typically this program is run from one of the many ARI start/stop methods and is not run stand-alone.

Stopping the Scheduler

ari_auto_scheduler.stop_scheduler removes the check_schedule procedure from the DBA_JOBS queue. This procedure must be run as the ARI master user to ensure that check_schedule is correctly removed. Typically this program is run from one of the many ARI start/stop methods and is not run stand-alone.

Signal-Driven Scheduler Signaler

This API can be added to other programs so that exception detection and/or event reevaluation can be attached to some other thing happening in the operational system. Each signal driven schedule has a signal text string. This string is passed in the appropriate argument of the ari_schedule_sql.accept_signal function to update the next execution date of the schedule associated with the signal text to the current date and time. This then causes processes associated with the schedule to be executed when the scheduler checker next executes.

> **Note:** This function conatins a commit, so likely placement of it in another procedure is just after a successful completion and commit of the procedure.

EVE (Exception Validation Engine)

EVE is the key threading process that governs ARI exception processing. It is typically started and stopped through one of the many ARI start/stop control methods, but the direct APIs for this functionality exist as well.

Starting EVE

ari_eve_process.start_eve adds the schedule_jobs procedure (EVE itself) to DBA_JOBS. This procedure must be run as the ARI master user to ensure that it has the appropriate privileges when it is executed by the DBA_JOBS queue manager process. Typically this program is run from one of the many ARI start/stop methods and is not run stand-alone.

Stopping EVE

ari_eve_process.stop_eve removes the schedule_jobs procedure (EVE itself) from the DBA_JOBS queue, and signals to the running process that it should terminate. The EVE process sends terminate signals to its Validator Threads and the EVE process self terminates. This procedure must be run as the ARI master user to ensure that it has the appropriate privileges when it executes in the queue. Typically this program is run from one of the many ARI start/stop methods and is not run stand-alone.

Periodic Purges

ARI uses a significant amount of data in two special areas, tracking events and tracking event history. Closed events that do not have a time-to-live (meaning almost all events – see the Online Help for reasons to create non-zero time-to-live events) are immediately removed from the system once they are closed. Events that do have a non-zero time-to-live must be periodically purged or they will remain in the system in a closed state indefinitely. Event History is retained after the event itself is closed for at least as long as it is specified in the event type definition, and is only removed by running the history purge process.

Event Purge

Because there will typically not be many events with a non-zero time-to-live, the need to periodically purge them is small, so the appropriate function ari_event_instance_sql.delete_expired need not be scheduled very often. On the other hand, because not many records will need purge, it should not take very long if run often, so it could equally be scheduled to run daily or weekly or even less often depending on the number and kinds of events in your system.

Event History Purge

Event history should probably be purged daily or at least weekly. Times to execute will vary based on volume of events, but presumably in full production the reason for setting a history interval is to somehow regulate table sizes. For this to work correctly the history tables themselves will need the periodic purging of event history that is older than the number of retention days specified in the event type definitions. The function to schedule for history purging is ari_event_hist_sql.purge_expired.

ARI Alert Notification API

The purpose of this API is to enable displaying a button on the toolbar of an application other than ARI itself. The ARI Alert Viewer button serves two functions: as a gateway for launching the alert viewer and as a visual indicator of the overall contents of the alert viewer. Launching the alert viewer simply means opening that Form. The visual indicator is that the button changes appearance depending on whether or not new alerts exist.

The API is implemented as a Forms library, a package and a table. The Forms library may not be appropriate if the application needing to interface with ARI is not a Forms application, but the substitution of that part of the API for a non-Forms application is an implementation issue not addressed by the product.

End User Cases

From the end-user standpoint, this API has the following cases:

- If the user is an ARI user with no new alerts, the button should be displayed on the toolbar with the mailbox closed icon (ari_ari0.ico).
- If the user is an ARI user with new alerts, the button should be displayed on the toolbar with the mailbox open icon (ari_ari6.ico).
- If the user is not an ARI user, the button is not displayed.
- In either scenario 1 or 2, the button should be pressed to attempt to launch the alert viewer. (This will display an error message if ARI is not truly installed.)
- When switching from either scenario 1 or 2 to scenario 3 (removing a user as an ARI user), the button will remain until the Form containing the button is closed and re-launched (which should give scenario 3). Attempting to launch the alert viewer from this button after being removed as a user will give an error. This seems annoying but is acceptable because removing an ARI user is very difficult unless the user has no events, and almost never happens in production unless a user has no events. Moreover, by simply exiting and re-launching the form the error is self-correcting.
- When switching from 3 to 1 or 2, the button will not appear until the Form containing the button is closed and re-launched.
- When switching between 1 and 2, the icon should change accordingly when focus is changed from the window and back again. Minimizing and then reselecting a window is one way to do this.

Architecture

ARI provides an API that gets installed in the external Forms application. This API is like a socket. The socket is only able to accept specific values and the ARI plug is constrained to only deliver those specific values.

ARI also provides a tester module that can be configured to simulate all three end-user cases so the external application can test application behavior for all potential plug outputs/ARI interactions. The tester configured to drive the third end-user case (so that no user is an ARI user) is essentially the safety cap mode that will be deployed if the external application, in spite of installing the socket, is never deployed with ARI. This is the socket that is already installed in Oracle Retail's Merchandise Transaction System as mentioned in the Process Details section of this document.

When ARI is implemented, the safety cap is replaced with a plug into ARI. This plug must have been tested in ARI to conform to the API specification that returns only three modes of operation corresponding to the first three end-user cases (the fourth case is an extension of one of the first two).

Implementation

Putting this all together, the socket part of the API is the Forms library. ariiflib.pll was installed in RMS. (That version is full compatible with the other components of the API and does not need to be replaced with the latest version ariif90.pll that is shipped with ARI for use with any other (non-RMS) applications.) The tester part is the stub version of the ari_interface_sql package and the ari_interface_test table. To plug this into ARI, the non-ARI application simply drops its ari_interface_sql package and uses a synonym to the ARI master schema version of this package instead.

Note: The localized modification of this package body for more sophisticated button icon display criteria, or entirely different return values (requires more API modification) is allowed in spite of statements elsewhere in this documentation that no modifications of this product are supported.

To install the socket the external application must add a button to the toolbar (or wherever they want a button) and must invoke the GET_IS_ARI_USER function to determine, based on whether the user is an ARI user, whether to display the button or enable its operation. This function should be installed so that the decision whether to display/enable the button is made with some frequency, depending on whether users are actively being added to and removed from the list of ARI users. Adding and removing users is non-trivial so the required frequency for such an action is minimal.

On some event-driven or periodic basis, including immediately after the button is displayed/enabled, the external application must invoke the REFRESH_ALERT_ICON function to update the button's displayed icon. The recommended implementation of this function is when the window focus changes (in the Forms When-Window-Activate trigger).

On pressing the alert viewer button the LAUNCH_ALERT_VIEWER procedure must be called. If ARI is installed on the same version of Forms as the application with which it will be working, this is a simple OPEN_FORM call. Otherwise, this is a remote call through the plug to tell ARI to launch, so the actual launching is external to the non-ARI application.

C

Appendix: ARI Options

The following ARI configuration options are on the ARI options table. These configuration options are critical settings that affect the performance and processing of ARI. However, once they are set, they can largely be ignored. These options are set by a database administrator using Oracle SQL*Plus doing simple option value updates based on option name.

EVE_NUM_THREADS

This is the number of realm queue processes, the validation engine should run at any one time. Generally this is the number of CPUs in the machine + 1, but not more than 32.

EVE_QUEUE_REFRESH_INTERVAL

This is the time slice in minutes given to each realm queue process. Ten minutes is recommended. It may need to be shorter to get reasonable real-time notification of exceptions if there are some very large queues and some very small ones. On the other hand, if the number of large queues is small enough, the entire time slice may only be used by one or two queues. If so, several processes will always be available for other processes, so a longer limit may improve the efficiency of processing those larger queues.

INTERNAL_SCHEMA

This is the name of the ARI user that is supposed to be created with the main ARI schema. This user has several explicit grants as described in the installation guide. It is in this schema that the ARI generated code and tables reside.

MASTER_SCHEMA

This is the main ARI schema. All of the static ARI code and tables are owned by this user, as described in the ARI Installation Guide.

MAX_EVENT_RECURSION

This value determines how many times an event will reevaluate itself as the result of auto-actions occurring before it raises an error. The value depends on the complexity of the states, an auto-action structure of the defined event types. Five is a reasonable value and no adjustment should be necessary except in the unlikely case of an event that needs to reevaluate legitimately (not though a mistake in the event definition) more than 5 times is created.

REEVAL_STATUS_LOCKOUT

This is the number of minutes between attempts to reevaluate events that could not be reevaluated either because they are locked or because their data values from remote data sources could not be fetched (e.g. because of a down database link).

PRIMARY_LANGUAGE_NUMBER

This is the number that identifies the language that will be considered the primary language for a multi-language ARI implementation. The primary language is also the only language in a single language implementation. The default value is based on the assumption that customers will use value 1 to represent their primary language.

ANALYST_ADMIN_GROUP_ID

This is here for reference by the ARI programs and should not be changed.

CLOSE_EVENT_REALM_ID

This is here for reference by the ARI programs and should not be changed unless the metadata for the close event action is altered by some mistake.

CLOSE_EVENT_REALM_ID

This is here for reference by the ARI programs and should not be changed unless the metadata for the close event action is altered by some mistake.

ERROR_ADMIN_GROUP_ID

This is here for reference by the ARI programs and should not be changed.

EVENT_INSTANCE_PARM_ID

This is here for reference by the ARI programs and should not be changed.

EXCEPTION_CREATE_DATE_PARM_ID

This is here for reference by the ARI programs and should not be changed.

EXCEPTION_CREATE_DATE_USER_ID

This is here for reference by the ARI programs and should not be changed.

LOG_LEVEL

Determines the log level of messages to be written to the ARI_ACTIVITY_LOG table. Valid levels are 1, 2 and 3. 3 is the most verbose logging and 1 only logs at a very high level. The default setting is 2, though this also may be unnecessarily verbose in a production environment.

FORWARD_GENERATION_HOURS

The value is number of hours and determines how far ahead the event and exception support code is generated before the definitions specify they are to become active. A detailed discussion exists in an earlier section on code generation. The default option is 30 hours allows some leeway if the code generator is schedule to run daily.

An alternative to routinely scheduled generation, supplemented by the occasional on-demand generation, might be to generate *only* at the business analyst's request, but the business analyst must then be careful that everything is a new start within the generation window or else it may not get generated. Starting something far in the future either requires a large generation window or a periodic generation so that, as the start time approaches, the analyst does not risk forgetting that the code generator (if it is not run periodically) needs to be run.

The downside of a large generation window is that, by generating far into the future, you cannot continue to modify a definition during the several days preceding its activation as you could with a 30-hour window and a daily code generation. To render such future-generated, no longer modifiable rules inactive, set the end date equal to the start data and regenerate code before the start date passes.

> **Note:** However, in production, modifications before deployment should not be necessary since all such issues should have been worked out in development.

Hence the definitions going into the production instance should have a solid foundation and a large generation window may be a perfectly fine approach. The issue is more significant in a development environment.