Oracle® Workflow

API Reference

Release 2.6.4

Part No. B15855-02

June 2005

ORACLE

Oracle Workflow API Reference, Release 2.6.4

Part No. B15855-02

Primary Author: Siu Chang, Clara Jaeckel

Contributing Author: Varsha Bhatia, George Buzsaki, John Cordes, Mark Craig, Avinash Dabholkar, Mark Fisher, Yongran Huang, Kevin Hudson, George Kellner, Sai Kilaru, Angela Kung, David Lam, Janet Lee, Jin Liu, Kenneth Ma, Steve Mayze, Santhana Natarajan, Rajesh Raheja, Varadarajan Rajaram, Tim Roveda, Robin Seiden, Vijay Shanmugam, Sachin Sharma, Sheryl Sheh, Allison Sparshott, Susan Stratton, Roshin Thomas, Robert Wunderlich

Contents

3 Directory Service APIs

4 Notification System APIs

5 Business Event System APIs

6 Workflow Queue APIs

7 Document Management APIs

Glossary

Index

Send Us Your Comments

Oracle Workflow API Reference, Release 2.6.4

Part No. B15855-02

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: appsdoc_us@oracle.com
- FAX: 650-506-7200 Attn: Oracle Applications Technology Group Documentation Manager
- Postal service:
  Oracle Applications Technology Group Documentation Manager
  Oracle Corporation
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Intended Audience

Welcome to Release 2.6.4 of the *Oracle Workflow API Reference*.

This guide assumes you have a working knowledge of the following:

- The principles and customary practices of your business area.
- Oracle Workflow.

  If you have never used Oracle Workflow, Oracle suggests you attend training classes available through Oracle University.
- The Oracle Applications graphical user interface.

  To learn more about the Oracle Applications graphical user interface, read the *Oracle Applications User's Guide*.
- Operating system concepts.
- Oracle Database, Oracle Application Server, and PL/SQL technology.

  If you have never used these products, Oracle suggests you attend training classes available through Oracle University.

See Related Documents on page xv for more Oracle Applications product information.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/ .

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Structure

Related Documents

You can choose from many sources of information, including online documentation, training, and support services, to increase your knowledge and understanding of Oracle Workflow.

If this guide refers you to other Oracle Applications documentation, use only the Release 11*i* versions of those guides.

Online Documentation

If you are using the version of Oracle Workflow embedded in Oracle Applications, note that all Oracle Applications documentation is available online (HTML or PDF).

- **PDF Documentation** - See the Oracle Applications Documentation Library CD for current PDF documentation for your product with each release. The Oracle Applications Documentation Library is also available on Oracle*MetaLink* and is updated frequently.
- **Online Help** - Online help patches (HTML) are available on Oracle*MetaLink*.
- **About Documents** - Refer to the About document for the mini-pack or family pack that you have installed to learn about feature updates, installation information, and new documentation or documentation patches that you can download. About documents are available on Oracle*MetaLink*.

If you are using the standalone version of Oracle Workflow, note that this guide is available online in HTML format. The HTML documentation is available from a URL provided by your system administrator or from the help icon in the Oracle Workflow Web pages.

Related Guides

You may want to refer to other Oracle Workflow guides and Oracle Applications implementation documentation when you set up and use Oracle Workflow. Additionally, Oracle Workflow is used by other Oracle Applications products to provide embedded workflows and business events. Therefore, if you are using the version of Oracle Workflow embedded in Oracle Applications, you may want to refer to other products' guides to learn more about the workflows and business events they include.

You can read the guides online by choosing Library from the expandable menu on your Oracle Applications HTML help window, by reading from the Oracle Applications Documentation Library CD included in your media pack, or by using a Web browser with a URL that your system administrator provides.

If you require printed guides, you can purchase them from the Oracle Store at http://oraclestore.oracle.com.

Guides Related to All Products

Oracle Applications User's Guide

This guide explains how to enter data, query, run reports, and navigate using the graphical user interface (GUI) available with this release of Oracle Workflow (and any other Oracle Applications products). This guide also includes information on setting user profiles, as well as running and reviewing reports and concurrent processes.

You can access this user's guide online by choosing "Getting Started with Oracle Applications" from any Oracle Applications help file.

Oracle Workflow Documentation Set

Oracle Workflow Administrator's Guide

This guide explains how to complete the setup steps necessary for any product that includes workflow-enabled processes, as well as how to monitor the progress of runtime workflow processes

Oracle Workflow Developer's Guide

This guide explains how to define new workflow business processes and customize existing Oracle Applications-embedded workflow processes. It also describes how to define and customize business events and event subscriptions.

Oracle Workflow User's Guide

This guide describes how users can view and respond to workflow notifications and monitor the progress of their workflow processes.

Guides Related to This Product

Oracle Assets User Guide

In Oracle Assets, you can post capital project costs to become depreciable fixed assets. Refer to this guide to learn how to query mass additions imported from other products to Oracle Assets and to review asset information.

Oracle General Ledger User Guide

Use this manual when you plan and define your chart of accounts, accounting period types and accounting calendar, functional currency, and set of books. The manual also describes how to define journal entry sources and categories so you can create journal entries for your general ledger. If you use multiple currencies, use this manual when you define additional rate types, and enter daily rates. This manual also includes complete information on implementing Budgetary Control.

Oracle HRMS Documentation Set

This set of guides explains how to define your employees, so you can give them operating unit and job assignments. It also explains how to set up an organization (operating unit). Even if you do not install Oracle HRMS, you can set up employees and organizations using Oracle HRMS windows. Specifically, the following manuals will help you set up employees and operating units:

- ***Using Oracle HRMS - The Fundamentals***

  This user guide explains how to set up and use enterprise modeling, organization management, and cost analysis.

- ***Managing People Using Oracle HRMS***

  Use this guide to learn about entering employees.

Oracle Payables User Guide

Refer to this manual to learn how to use Invoice Import to create invoices in Oracle Payables from expense reports data in the Oracle Payables interface tables. This manual

also explains how to define suppliers, and how to specify supplier and employee numbering schemes for invoices.

Oracle Projects Implementation Guide

Use this manual as a guide for implementing Oracle Projects. This manual also includes appendixes covering function security, menus and responsibilities, and profile options.

Oracle Purchasing User Guide

Use this guide to learn about entering and managing the requisitions and purchase orders that relate to your projects. This manual also explains how to create purchase orders from project-related requisitions in the AutoCreate Documents window.

Oracle Receivables User Guide

Use this manual to learn more about Oracle Receivables invoice processing and invoice formatting, defining customers, importing transactions using AutoInvoice, and defining automatic accounting in Oracle Receivables.

Oracle Business Intelligence System Implementation Guide

This guide provides information about implementing Oracle Business Intelligence (BIS) in your environment

BIS 11i User Guide Online Help

This guide is provided as online help only from the BIS application and includes information about intelligence reports, Discoverer workbooks, and the Performance Management Framework.

Using Oracle Time Management

This guide provides information about capturing work patterns such as shift hours so that this information can be used by other applications such as Oracle General Ledger.

Installation and System Administration

Oracle Applications Concepts

This guide provides an introduction to the concepts, features, technology stack, architecture, and terminology for Oracle Applications Release 11*i*. It provides a useful first book to read before installing Oracle Applications.

Installing Oracle Applications

This guide provides instructions for managing the installation of Oracle Applications products. In Release 11*i*, much of the installation process is handled using Oracle Rapid Install, which minimizes the time to install Oracle Applications and the Oracle technology stack by automating many of the required steps. This guide contains instructions for using Oracle Rapid Install and lists the tasks you need to perform to finish your installation. You should use this guide in conjunction with individual product user's guides and implementation guides.

Upgrading Oracle Applications

Refer to this guide if you are upgrading your Oracle Applications Release 10.7 or Release 11.0 products to Release 11*i*. This guide describes the upgrade process and lists database and product-specific upgrade tasks. You must be either at Release 10.7 (NCA, SmartClient, or character mode) or Release 11.0 to upgrade to Release 11*i*. You cannot upgrade to Release 11*i* directly from releases prior to 10.7.

Maintaining Oracle Applications

Use this guide to help you run the various AD utilities, such as AutoUpgrade, Auto Patch, AD Administration, AD Controller, AD Relink, License Manager, and others. It contains how-to steps, screenshots, and other information that you need to run the AD utilities. This guide also provides information on maintaining the Oracle Applications file system and database.

Oracle Applications System Administrator's Guide

This guide provides planning and reference information for the Oracle Applications system administrator. It contains information on how to define security, customize menus and online help, and manage concurrent processing.

Oracle Alert User's Guide

This guide explains how to define periodic and event alerts to monitor the status of your Oracle Applications data.

Other Implementation Documentation

Oracle Applications Product Update Notes

Use this guide as a reference for upgrading an installation of Oracle Applications. It provides a history of the changes to individual Oracle Applications products between Release 11.0 and Release 11*i*. It includes new features, enhancements, and changes made to database objects, profile options, and seed data for this interval.

Multiple Reporting Currencies in Oracle Applications

If you use the Multiple Reporting Currencies feature to record transactions in more than one currency, use this manual before implementing Oracle Applications. This manual details additional steps and setup considerations for implementing Oracle Applications with this feature.

Multiple Organizations in Oracle Applications

This guide describes how to set up and use Oracle Applications' Multiple Organization support feature, so you can define and support different organization structures when running a single installation of Oracle Applications.

Oracle Applications Flexfields Guide

This guide provides flexfields planning, setup, and reference information for the Oracle Applications implementation team, as well as for users responsible for the ongoing maintenance of Oracle Applications product data. This guide also provides information on creating custom reports on flexfields data.

Oracle Applications Developer's Guide

This guide contains the coding standards followed by the Oracle Applications development staff. It describes the Oracle Application Object Library components needed to implement the Oracle Applications user interface described in the *Oracle Applications User Interface Standards for Forms-Based Products*. It also provides information to help you build your custom Oracle Forms Developer forms so that they integrate with Oracle Applications.

Oracle Applications User Interface Standards for Forms-Based Products

This guide contains the user interface (UI) standards followed by the Oracle Applications development staff. It describes the UI for the forms-based Oracle Applications products and how to apply this UI to the design of an application built using Oracle Forms.

Oracle eTechnical Reference Manuals

Each eTechnical Reference Manual (eTRM) contains database diagrams and a detailed description of database tables, forms, reports, and programs for a specific Oracle Applications product. This information helps you convert data from your existing applications, integrate Oracle Applications data with non-Oracle applications, and write custom reports for Oracle Applications products. Oracle eTRM is available on Oracle*MetaLink*.

Oracle Applications Message Reference Manual

This manual describes Oracle Applications messages. This manual is available in HTML format on the documentation CD-ROM for Release 11*i*.

Training and Support

Training

Oracle offers a complete set of training courses to help you and your staff master Oracle Workflow and reach full productivity quickly. These courses are organized into functional learning paths, so you take only those courses appropriate to your job or area of responsibility.

You have a choice of educational environments. You can attend courses offered by Oracle University at any one of our many Education Centers, you can arrange for our trainers to teach at your facility, or you can use Oracle Learning Network (OLN), Oracle University's online education utility. In addition, Oracle training professionals can tailor standard courses or develop custom courses to meet your needs. For example, you may want to use your organization's structure, terminology, and data as examples in a customized training session delivered at your own facility.

Support

From on-site support to central support, our team of experienced professionals provides the help and information you need to keep Oracle Workflow working for you. This team includes your Technical Representative, Account Manager, and Oracle's large staff of consultants and support specialists, with expertise in your business area, managing an Oracle Database, and your hardware and software environment.

Do Not Use Database Tools to Modify Oracle Applications Data

Oracle STRONGLY RECOMMENDS that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle Applications data unless otherwise instructed.

Oracle provides powerful tools you can use to create, store, change, retrieve, and maintain information in an Oracle database. But if you use Oracle tools such as SQL*Plus to modify Oracle Applications data, you risk destroying the integrity of your data and you lose the ability to audit changes to your data.

Because Oracle Applications tables are interrelated, any change you make using an Oracle Applications form can update many tables at once. But when you modify Oracle Applications data using anything other than Oracle Applications, you may change a row in one table without making corresponding changes in related tables. If your tables get

out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle Applications.

When you use Oracle Applications to modify your data, Oracle Applications automatically checks that your changes are valid. Oracle Applications also keeps track of who changes information. If you enter information into database tables using database tools, you may store invalid information. You also lose the ability to track who has changed your information because SQL*Plus and other database tools do not keep a record of changes.

1

Overview of Oracle Workflow

This chapter introduces you to the concept of a workflow process and to the major features of Oracle Workflow.

This chapter covers the following topics:

- Overview of Oracle Workflow
- Oracle Workflow Procedures and Functions

Overview of Oracle Workflow

Oracle Workflow delivers a complete workflow management system that supports business process based integration. Its technology enables modeling, automation, and continuous improvement of business processes, routing information of any type according to user-defined business rules.

E-business is accelerating the demand for integration of applications within the enterprise as well as integration of a company's systems with trading partners and business-to-business exchanges. Oracle Workflow automates and streamlines business processes both within and beyond your enterprise, supporting traditional applications based workflow as well as e-business integration workflow. Oracle Workflow is unique in providing a workflow solution for both internal processes and business process coordination between applications.

Routing Information

Business processes today involve getting many types of information to multiple people according to rules that are constantly changing. With so much information available, and in so many different forms, how do you get the right information to the right people? Oracle Workflow lets you provide each person with all the information they need to take action. Oracle Workflow can route supporting information to each decision maker in a business process, including people both inside and outside your enterprise.

Defining and Modifying Business Rules

Oracle Workflow lets you define and continuously improve your business processes using a drag-and-drop process designer.

Unlike workflow systems that simply route documents from one user to another with some approval steps, Oracle Workflow lets you model sophisticated business processes. You can define processes that loop, branch into parallel flows and then rendezvous, decompose into subflows, and more. Because Oracle Workflow can decide which path to take based on the result of a stored procedure, you can use the power of

Java and of PL/SQL, the language of the Oracle Database, to express any business rule that affects a workflow process. See: Workflow Processes, page 1-4.

Delivering Electronic Notifications

Oracle Workflow extends the reach of business process automation throughout the enterprise and beyond to include any e-mail or Internet user. Oracle Workflow lets people receive notifications of items awaiting their attention via e-mail, and act based on their e-mail responses. You can even view your list of things to do, including necessary supporting information, and take action using a standard Web browser.

Integrating Systems

Oracle Workflow lets you set up subscriptions to business events which can launch workflows or enable messages to be propagated from one system to another when business events occur. You can communicate events among systems within your own enterprise and with external systems as well. In this way, you can implement point-to-point messaging integration or use Oracle Workflow as a messaging hub for more complex system integration scenarios. You can model business processes that include complex routing and processing rules to handle events powerfully and flexibly.

Major Features and Definitions

Oracle Workflow Builder

Oracle Workflow Builder is a graphical tool that lets you create, view, or modify a business process with simple drag and drop operations. Using the Workflow Builder, you can create and modify all workflow objects, including activities, item types, and messages. See: Workflow Processes, page 1-4.

At any time you can add, remove, or change workflow activities, or set up new prerequisite relationships among activities. You can easily work with a summary-level model of your workflow, expanding activities within the workflow as needed to greater levels of detail. And, you can operate Oracle Workflow Builder from a desktop PC or from a disconnected laptop PC.

Workflow Engine

The Workflow Engine embedded in the Oracle Database implements process definitions at runtime. The Workflow Engine monitors workflow states and coordinates the routing of activities for a process. Changes in workflow state, such as the completion of workflow activities, are signaled to the engine via a PL/SQL API or a Java API. Based on flexibly-defined workflow rules, the engine determines which activities are eligible to run, and then runs them. The Workflow Engine supports sophisticated workflow rules, including looping, branching, parallel flows, and subflows.

Business Event System

The Business Event System is an application service that uses the Oracle Advanced Queuing (AQ) infrastructure to communicate business events between systems. The Business Event System consists of the Event Manager, which lets you register subscriptions to significant events, and event activities, which let you model business events within workflow processes.

When a local event occurs, the subscribing code is executed in the same transaction as the code that raised the event. Subscription processing can include executing custom code on the event information, sending event information to a workflow process, and sending event information to other queues or systems.

Workflow Definitions Loader

The Workflow Definitions Loader is a utility program that moves workflow definitions between database and corresponding flat file representations. You can use it to move workflow definitions from a development to a production database, or to apply upgrades to existing definitions. In addition to being a standalone server program, the Workflow Definitions Loader is also integrated into Oracle Workflow Builder, allowing you to open and save workflow definitions in both a database and file.

Complete Programmatic Extensibility

Oracle Workflow lets you include your own PL/SQL procedures or external functions as activities in your workflows. Without modifying your application code, you can have your own program run whenever the Workflow Engine detects that your program's prerequisites are satisfied.

Electronic Notifications

Oracle Workflow lets you include users in your workflows to handle activities that cannot be automated, such as approvals for requisitions or sales orders. The Notification System sends notifications to and processes responses from users in a workflow. Electronic notifications are routed to a role, which can be an individual user or a group of users. Any user associated with that role can act on the notification.

Each notification includes a message that contains all the information a user needs to make a decision. The information may be embedded in the message body or attached as a separate document. Oracle Workflow interprets each notification activity response to decide how to move on to the next workflow activity.

Electronic Mail Integration

Electronic mail (e-mail) users can receive notifications of outstanding work items and can respond to those notifications using their e-mail application of choice. An e-mail notification can include an attachment that provides another means of responding to the notification.

Internet-Enabled Workflow

Any user with access to a standard Web browser can be included in a workflow. Web users can access a Notification Web page to see their outstanding work items, then navigate to additional pages to see more details or provide a response.

Monitoring and Administration

Workflow administrators and users can view the progress of a work item in a workflow process by connecting to the Workflow Monitor using a standard Web browser that supports Java. The Workflow Monitor displays an annotated view of the process diagram for a particular instance of a workflow process, so that users can get a graphical depiction of their work item status. The Workflow Monitor also displays a separate status summary for the work item, the process, and each activity in the process.

If you are using the version of Oracle Workflow embedded in Oracle Applications and you have implemented Oracle Applications Manager, you can also use the Oracle Workflow Manager component of Oracle Applications Manager as an additional administration tool for Oracle Workflow. Oracle Applications Manager is a tool that provides administrative and diagnostic capabilities for concurrent processing, Oracle Workflow, and other functionality in Oracle Applications. For more information, please refer to the Oracle Applications Manager online help.

Also, if you are using the standalone version of Oracle Workflow, you can use the standalone Oracle Workflow Manager component available through Oracle Enterprise Manager as an additional administration tool for Oracle Workflow. For more information, please refer to the Oracle Workflow Manager online help.

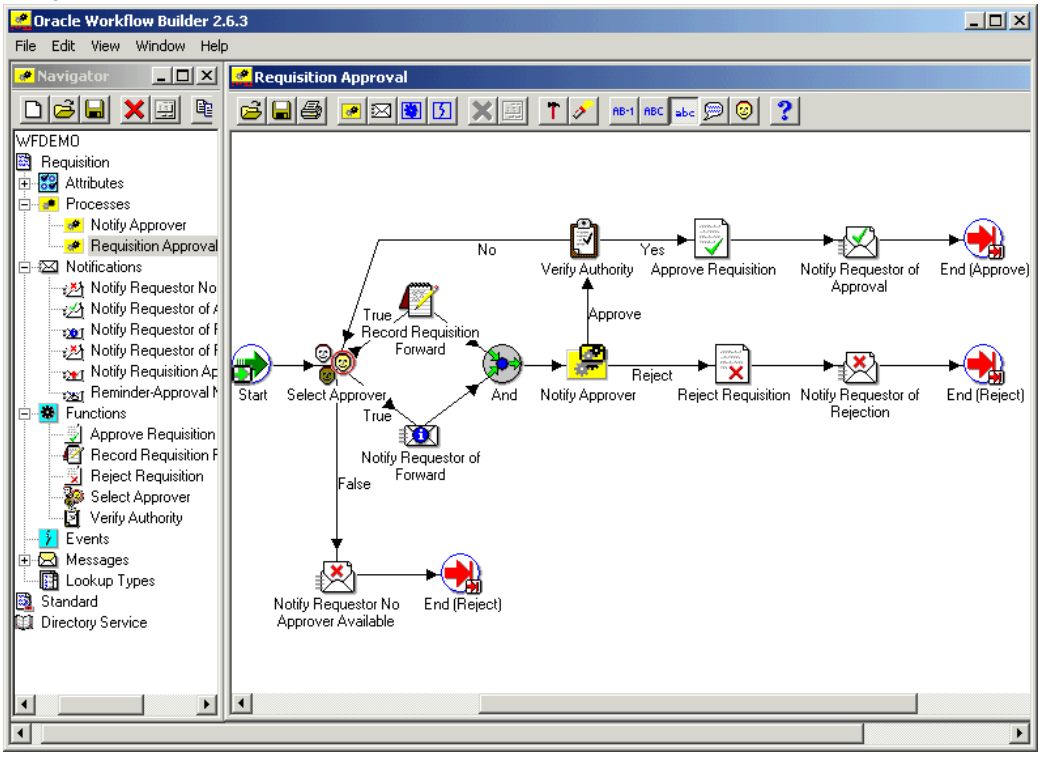
Workflow Processes

Oracle Workflow manages business processes according to rules that you define. The rules, which we call a workflow process definition, include the activities that occur in the process and the relationship between those activities. An activity in a process definition can be an automated function defined by a PL/SQL stored procedure or an external function, a notification to a user or role that may optionally request a response, a business event, or a subflow that itself is made up of a more granular set of activities.

A workflow process is initiated when an application calls a set of Oracle Workflow Engine APIs. The Workflow Engine takes over by driving the relevant work item defined by the application, through a specific workflow process definition. According to the workflow process definition, the Workflow Engine performs automated steps and invokes appropriate agents when external processing is required.

The following diagram depicts a simplified workflow process definition that routes a requisition to a manager or set of managers for approval.

Sample Workflow Process in Oracle Workflow Builder



We refer to the whole drawing as a process or process diagram. The icons represent activities, and the arrows represent the transitions between the activities. In the above example, new items are created for the process when a user creates and submits a requisition in the appropriate application.

This process contains several workflow activities implemented as PL/SQL stored procedures, including:

- Select Approver - to select, according to your business rules, who should approve the requisition.
- Verify Authority - to verify that a selected approver has the spending authority to approve the requisition.

Oracle Workflow Procedures and Functions

Oracle Workflow supplies a list of public PL/SQL and Java procedures and functions that you can use to set up a workflow process. They are grouped within the following packages and classes:

- WF_ENGINE, page 2-14
- WFFunctionAPI, page 2-60
- WFAttribute, page 2-64
- WF_CORE, page 2-68
- WF_PURGE, page 2-76
- WF_MONITOR, page 2-84

- WF_FWKMON, page 2-90
- Oracle Workflow Views, page 2-91
- WF_DIRECTORY, page 3-1
- WF_LDAP, page 3-18
- WF_LOCAL_SYNCH, page 3-21
- WF_ROLE_HIERARCHY, page 3-30
- WF_PREF, page 3-33
- WF_NOTIFICATIONS, page 4-14
- WF_MAIL_UTIL, page 4-44
- WF_EVENT, page 5-20
- WF_RULE, page 5-33
- WF_EVENT_FUNCTIONS_PKG, page 5-44
- WF_EVENTS_PKG, page 5-50
- WF_EVENT_GROUPS_PKG, page 5-50
- WF_SYSTEMS_PKG, page 5-50
- WF_AGENTS_PKG, page 5-50
- WF_AGENT_GROUPS_PKG, page 5-50
- WF_EVENT_SUBSCRIPTIONS_PKG, page 5-50
- WF_BES_CLEANUP, page 5-58
- WF_QUEUE, page 6-1
- FND_DOCUMENT_MANAGEMENT, page 7-1

2

Workflow Engine APIs

This chapter describes the APIs for the Workflow Engine. The APIs consist of views and PL/SQL and Java functions and procedures that you can use to access the Workflow Engine, the Workflow Monitor, and workflow data.

This chapter covers the following topics:

- Overview of the Workflow Engine
- Workflow Engine APIs
- Workflow Function APIs
- Workflow Attribute APIs
- Workflow Core APIs
- Workflow Purge APIs
- Workflow Monitor APIs
- Workflow Status Monitor APIs
- Oracle Workflow Views

Overview of the Workflow Engine

The Workflow Engine manages all automated aspects of a workflow process for each item. The engine is implemented in server-side PL/SQL and is activated whenever a call to a workflow procedure or function is made. Since the engine is embedded inside the Oracle Database, if the Workflow server goes down for any reason, the Oracle Database is able to manage the recovery and transactional integrity of any workflow transactions that were running at the time of the failure.

Additionally, Workflow engines can be set up as background tasks to perform activities that are too costly to execute in real time.

The Workflow Engine performs the following services for a client application:

- It manages the state of all activities for an item, and in particular, determines which new activity to transition to whenever a prerequisite activity completes.
- It automatically executes function activities (execution is either immediate or deferred to a background engine) and sends notifications.
- It maintains a history of an activity's status.
- It detects error conditions and executes error processes.

The state of a workflow item is defined by the various states of all activities that are part of the process for that item. The engine changes activity states in response to an API call to update the activity. The API calls that update activity states are:

- CreateProcess, page 2-16
- StartProcess, page 2-21
- CompleteActivity, page 2-50
- CompleteActivityInternalName, page 2-52
- AssignActivity, page 2-53
- HandleError, page 2-55
- SuspendProcess, page 2-24
- ResumeProcess, page 2-25
- AbortProcess, page 2-26

Based on the result of a previous activity, the engine attempts to execute the next activity directly. An activity may have the following status:

- Active - activity is running.
- Complete - activity completed normally.
- Waiting - activity is waiting to run.
- Notified - notification activity is delivered and open.
- Deferred - activity is deferred.
- Error - activity completed with error.
- Suspended - activity is suspended.

Important: The Workflow Engine traps errors produced by function activities by setting a savepoint before each function activity. If an activity produces an unhandled exception, the engine performs a rollback to the savepoint, and sets the activity to the `ERROR` status. For this reason, you should never commit within the PL/SQL procedure of a function activity. The Workflow Engine never issues a commit as it is the responsibility of the calling application to commit.

For environments such as database triggers or distributed transactions that do not allow savepoints, the Workflow Engine automatically traps "Savepoint not allowed" errors and defers the execution of the activity to the background engine.

Note: The Oracle Database supports autonomous transactions. By embedding the pragma `AUTONOMOUS_TRANSACTION` in your procedure, you can perform commits and rollbacks independently of the main transaction. Oracle treats this as a separate session; as such, you will not have access to any database changes that were made in the main session but are not yet committed. Consequently, you are restricted from updating workflow-specific data in an autonomous transaction; for instance, you cannot set item attributes. You cannot access this data because the item itself has not yet been committed, and because you may have lock contentions with the main session.

Oracle Workflow will not support autonomous commits in any procedure it calls directly. If you need to perform commits, then embed your SQL in a subprocedure and declare it as an autonomous block. This subprocedure must be capable of being rerun. Additionally, note that Oracle Workflow handles errors by rolling back the entire procedure and setting its status to `ERROR`. Database updates performed by autonomous commits cannot be rolled back, so you will need to write your own compensatory logic for error handling. For more information, see: Autonomous Transactions, *Oracle Database Concepts*.

Oracle Workflow Java Interface

The Oracle Workflow Java interface provides a means for any Java program to integrate with Oracle Workflow. The Oracle Workflow Engine and Notification APIs are accessible through public server PL/SQL packages and published views. The Oracle Workflow Java interface exposes those APIs as Java methods that can be called by any Java program to communicate with Oracle Workflow. The Java methods directly reference the WF_ENGINE and WF_NOTIFICATION PL/SQL package procedures and views and communicate with the Oracle Workflow database through JDBC.

The methods are defined within the WFEngineAPI class and the WFNotificationAPI class, in the Java package '`oracle.apps.fnd.wf.engine`'. If a Workflow Engine or notification API has a corresponding Java method, its Java method syntax is displayed immediately after its PL/SQL syntax in the documentation. See: Workflow Engine APIs, page 2-14 and Notification APIs, page 4-14.

Additionally, Java functions can be incorporated within Workflow processes as external Java function activities. This functionality is currently only available for the standalone version of Oracle Workflow. The custom Java classes for these activities are implemented as classes that extend the WFFunctionAPI class. The custom classes must follow a standard API format so that they can be properly executed by the Oracle Workflow Java Function Activity Agent. See: Standard API for Java Procedures Called by Function Activities, *Oracle Workflow Developer's Guide* and Function Activity, *Oracle Workflow Developer's Guide*.

The WFFunctionAPI class and the WFAttribute class also contain methods that can be called to communicate with Oracle Workflow. These classes are defined in the Java package '`oracle.apps.fnd.wf`'. See: Workflow Function APIs, page 2-60 and Workflow Attribute APIs, page 2-64.

Java programs that integrate with Oracle Workflow should include the following import statements to provide access to classes required by Oracle Workflow:

```
import java.io.*;
import java.sql.*;
import java.math.BigDecimal;

import oracle.sql.*;
import oracle.jdbc.driver.*;

import oracle.apps.fnd.common.*;
import oracle.apps.fnd.wf.engine.*;
import oracle.apps.fnd.wf.*;
```

Oracle Workflow Context

Each Oracle Workflow Java method that accesses the database requires an input of a WFContext object. The WFContext object consists of database connectivity information which you instantiate and resource context information that the WFContext class instantiates. To call one of these Workflow Java APIs in your Java program, you must first instantiate a database variable of class WFDB with your database username, password and alias. You can also optionally supply a JDBC string. Then you must instantiate the WFContext object with the database variable. You can retrieve the system property CHARSET to specify the character set for the database session. The following code excerpt shows an example of how to instantiate these objects.

```
WFDB myDB;
WFContext ctx;

myDB = new WFDB(m_user, m_pwd, m_jdbcStr, m_conStr);
m_charSet = System.getProperty("CHARSET");
if (m_charSet == null) { // cannot be null
  m_charSet = "UTF8";

}

try {
  ctx = new WFContext(myDB, m_charSet);
  // m_charSet is 'UTF8' by default

  if (ctx.getDB().getConnection() == null) {
    // connection failed
    return;
  }

  // We now have a connection to the database.
}

catch (Exception e) {
// exit Message for this exception
}
```

If you have already established a JDBC connection, you can simply set that connection into the WFContext object, as shown in the following example:

```
WFContext ctx;

m_charSet = System.getProperty("CHARSET");
if (m_charSet == null) { // cannot be null
  m_charSet = "UTF8";
}

ctx = new WFContext(m_charSet);
// m_charSet is 'UTF8' by default

ctx.setJDBCConnection(m_conn);
// m_conn is a pre-established JDBC connection
```

The Oracle Workflow Java APIs can be used safely in a thread, with certain restrictions:

- Each thread should have its own WFContext object, meaning you should not instantiate WFContext before starting threads. Because each context keeps track of an error stack, contexts cannot be shared.
- You should not use the same JDBC connection for different workflows, because using the same connection may cause problems with commits and rollbacks for unrelated transactions.

There is no synchronized code inside the Oracle Workflow Java APIs, but there are no shared resources, either.

There is also no connection pooling in the Oracle Workflow Java APIs. For Oracle Applications, connection pooling is implemented at the AOL/J level; after you get the JDBC connection, you use the *WFContext.setJDBCConnection()* API to set the connection. This approach lets you manage your JDBC connection outside of the Oracle Workflow APIs.

Sample Java Program

Oracle Workflow provides an example Java program that illustrates how to call most of the Workflow Engine and Notification Java APIs. The Java program is named WFTest. It calls the various Java APIs to launch the WFDEMO process, set and get attributes, and suspend, resume, and abort the process, as well as the APIs to send a notification, set and get notification attributes, and delegate and transfer the notification. Before running the WFTest Java program, make sure you define `CLASSPATH` and `LD_LIBRARY_PATH` for the Oracle JDBC implementation and a supported version of Oracle. For example, on UNIX, use the following commands:

```
setenv CLASSPATH
<Workflow_JAR_file_directory>/wfapi.jar:${ORACLE_HOME}/jdbc/lib/cl
asses111.zip

setenv LD_LIBRARY_PATH ${ORACLE_HOME}/lib:${LD_LIBRARY_PATH}
```

> **Note:** If you are using the standalone version of Oracle Workflow, the Workflow JAR files are located in the *<ORACLE_HOME>*`/jlib` directory. If you are using the version of Oracle Workflow embedded in Oracle Applications, the Workflow JAR files are located in the *<ORACLE_HOME>*`/wf/java/oracle/apps/fnd/wf/jar/` directory.

To initiate the WFTest program, run Java against `oracle.apps.fnd.wf.WFTest`. For example, on UNIX, enter the following statement on the command line:

```
$java oracle.apps.fnd.wf.WFTest
```

The source file for this program is also included in your Oracle Workflow installation so that you can view the sample code. The source file is named `WFTest.java` and is located in the *<ORACLE_HOME>*`/wf/java/oracle/apps/fnd/wf/` directory.

Additional Workflow Engine Features

In addition to managing a process, the Workflow Engine also supports the following features:

- Completion Processing, page 2-6
- Deferred Processing, page 2-6

Completion Processing

Engine processing is triggered whenever a process activity completes and calls the Workflow Engine API. The engine then attempts to execute (or mark for deferred execution) all activities that are dependent on the completed activity.

> **Note:** A process as a whole can complete but still contain activities that were visited but not yet completed. For example, a completed process may contain a standard Wait activity that is not complete because the designated length of time to wait has not yet elapsed. When the process as a whole completes, the Workflow Engine marks these incomplete activities as having a status of `COMPLETE` and a result of `#FORCE`. This distinction is important when you review your process status through the Workflow Monitor.

Deferred Processing

The engine has a deferred processing feature that allows long-running tasks to be handled by background engines instead of in real time. Deferring the execution of activity functions to background engines allows the Workflow Engine to move forward to process other activities that are currently active. The engine can be set up to operate anywhere on a continuum between processing all eligible work immediately, to processing nothing and marking all transitions as deferred.

Each activity has a user-defined processing cost. You can set this cost to be small if the activity merely sets an item attribute, or you may set it to be very high if the activity performs a resource-intensive operation. If the result of a completed activity triggers the execution of a costly function, you might want to defer the execution of that costly function to a background engine.

The Workflow Engine integrates with Oracle Advanced Queues to carry out deferred processing. If a function activity has a cost that exceeds the main threshold cost, the Workflow Engine marks that activity with a status of '`DEFERRED`' in the workflow status tables and enqueues the deferred activity to a special queue for deferred activities. A special queue processor called the background engine checks and processes the activities in the 'deferred' queue. The order in which the deferred activities are processed are based on the first in, first out ordering of an activity's enqueue time. At least one background engine must be set up to run at all times. Some sites may have multiple background engines operating at different thresholds or item type specifications, to avoid tying up all background processing with long-running operations.

See: Setting Up Background Engines, *Oracle Workflow Administrator's Guide*, Activity Cost, *Oracle Workflow Developer's Guide*, and Deferring Activities, *Oracle Workflow Administrator's Guide*.

Error Processing

Errors that occur during workflow execution cannot be directly returned to the caller, since the caller generally does not know how to respond to the error (in fact, the caller may be a background engine with no human operator). You can use Oracle Workflow Builder to define the processing you want to occur in case of an error. Use Oracle Workflow Builder to assign the Default Error Process associated with the System:Error item type or create your own custom error process. See: Error Handling for Workflow Processes, *Oracle Workflow Developer's Guide*.

The error process can include branches based on error codes, send notifications, and attempt to deal with the error using automated rules for resetting, retrying, or skipping the failed activity. Once you define an error process, you can associate it with any activity. The error process is then initiated whenever an error occurs for that activity. See: To Define Optional Activity Details, *Oracle Workflow Developer's Guide*.

The Workflow Engine traps errors produced by function activities by setting a savepoint before each function activity. If an activity produces an unhandled exception, the engine performs a rollback to the savepoint, and sets the activity to the `ERROR` status.

> **Note:** For this reason, you should never commit within the PL/SQL procedure of a function activity. The Workflow Engine never issues a commit as it is the responsibility of the calling application to commit.

The Workflow Engine then attempts to locate an error process to run by starting with the activity which caused the error, and then checking each parent process activity until an associated error process is located.

Looping

Looping occurs when the completion of an activity causes a transition to another activity that has already been completed. The first activity that gets detected as a revisited activity is also called a loop point or pivot activity. The Workflow Engine can handle a revisited activity in one of three ways:

- Ignore the activity, and stop further processing of the thread, so in effect, the activity can only run once.
- Reset the loop to the loop point before reexecuting by first running logic to undo the activities within the loop.
- Reexecute the loop point and all activities within the loop without running any logic.

Every activity has an On Revisit poplist field in its Oracle Workflow Builder Details property page. The On Revisit poplist lets you specify the behavior of the Workflow Engine when it revisits the activity in a workflow process. You can set the field to `Ignore`, `Reset`, or `Loop`.

Setting On Revisit to `Ignore` is useful for implementing activities that should only run once, even though they can be transitioned to from multiple sources. For example, this mode allows you to implement a "logical OR" type of activity which is transitioned to multiple times, but completes after the first transition only.

Setting On Revisit to `Reset` for an activity is useful when you want to reexecute activities in a loop, but you want to first reset the status of the activities in the loop. Reset causes the Workflow Engine to do the following:

- Build a list of all activities visited following the pivot activity.

- Traverse the list of activities, cancelling each activity and resetting its status.

Cancelling an activity is similar to executing the activity, except that the activity is executed in "CANCEL" mode rather than "RUN" mode. You can include compensatory logic in "CANCEL" mode that reverses any operation performed earlier in "RUN" mode.

If you set On Revisit to Reset for the pivot activity of a loop that includes an FYI notification activity, the Workflow Engine cancels the previous notification before reexecuting the loop and sending a new notification to the current performer of the notification activity.

Setting On Revisit to Loop for an activity is useful when you want to simply reexecute activities in a loop without resetting the status of the activities in the loop. Loop causes the Workflow Engine to reexecute the activity in "RUN" mode without executing any "CANCEL" mode logic for the activity.

If you set On Revisit to Loop for the pivot activity of a loop that includes an FYI notification activity, previous notifications remain open when the Workflow Engine reexecutes the loop and sends a new notification to the current performer of the notification activity.

Version / Effective Date

Certain workflow objects in a process definition are marked with a version number so that more than one version of the object can be in use at any one time. These objects are:

- Activities - notifications, functions, and processes

  Note: Although function activities support versioning, the underlying PL/SQL code does not, unless implemented by your developer. You should avoid adding references to new activity attributes or returning result lookup codes not modelled by existing activities in your PL/SQL code.

- Activity attributes
- Process activity nodes
- Activity attribute values
- Activity transitions

If you edit and save any of the above objects in Oracle Workflow Builder to the database, Oracle Workflow automatically creates a new version of that object or the owning object by incrementing the version number by one. If you save edits to any of the above objects to an existing file, then the original objects are overwritten. If you have a process instance that is still running and you upgrade the underlying workflow definition in your Workflow server, the process instance continues to run using the version of the workflow object definitions with which it was originally initiated.

An effective date controls which version of a definition the engine uses when executing a process. When you edit a process, you can save it with an immediate or future effective date. Any new process instance that is initiated always uses the version that is specified to be effective at that point in time. See: Opening and Saving Item Types, *Oracle Workflow Developer's Guide*.

Note that Oracle Workflow does not maintain versions for other workflow objects. Any modifications that you save to the following objects overwrites the existing definition of the object:

- Item attributes
- Messages
- Lookup types

Item Type Attributes

A set of item type attributes is defined at both design-time and runtime for each item. These attributes provide information to the function and notification activities used in the processes associated with the item type.

When you define item type attributes at runtime, you can add either individual attributes or arrays containing several attributes of the same type, using the appropriate Workflow Engine APIs. Similarly, you can set the values of existing attributes either individually or in arrays containing several attributes of the same type.

Use the array APIs whenever you need to add or set the values of large numbers of item type attributes at once. These APIs improve performance by using the bulk binding feature in the Oracle Database to reduce the number of database operations. See: AddItemAttributeArray, page 2-34 and SetItemAttributeArray, page 2-39.

> **Note:** These array APIs handle arrays that are composed of multiple item type attributes grouped together by type. Oracle Workflow does not support individual item type attributes that consist of arrays themselves.

Post-Notification Functions

You can associate a post-notification function with a notification activity. The Workflow Engine executes the post-notification function in response to an update of the notification's state after the notification is delivered. For example, you can specify a post-notification function that executes when the notification recipient forwards or transfers the notification. The post-notification function could perform back-end logic to either validate the legitimacy of the forward or transfer or execute some other supporting logic.

The post-notification function should be a PL/SQL procedure written to the same API standards required for function activities. See: Standard API for PL/SQL Procedures Called by Function Activities, *Oracle Workflow Developer's Guide*.

When you specify a post-notification function, the Workflow Engine first sets the context information to use with the function through the following global engine variables. In some cases the values of the variables differ depending on the mode in which the post-notification function is called.

- `WF_ENGINE.context_nid` - The notification ID. For `RUN` or `TIMEOUT` mode, if the Expand Roles property is checked for the notification activity, then this variable contains the notification group ID for the notifications sent to the individual members of the role.
- `WF_ENGINE.context_user` - The user who is responsible for taking the action that updated the notification's state.
  - For `RESPOND`, `FORWARD`, `TRANSFER`, `QUESTION`, or `ANSWER` mode, if the user was acting on his or her own behalf, then the value of `WF_ENGINE.context_user` varies depending on the notification interface. If the user acted through the Notification Details Web page, then `WF_ENGINE.context_user` is set to the

user name of the logged in user. If the recipient acted through e-mail, then this variable is set to 'email:'*<email_address>*.

- For RESPOND, FORWARD, TRANSFER, QUESTION, or ANSWER mode, if the user was acting on behalf of another user by accessing that user's Worklist Web page through the worklist access feature, then WF_ENGINE.context_user is set to the user name of that other user, to whom that worklist belongs.
- For RUN or TIMEOUT mode, WF_ENGINE.context_user is set to the role assigned as the performer of the notification activity.

- WF_ENGINE.context_user_comment - Comments appended to the notification.
  - For RESPOND mode, this variable is set to any comments entered in the special WF_NOTE Respond message attribute, if that attribute is defined for the notification.
  - For FORWARD or TRANSFER mode, this variable is set to any comments entered when the notification was reassigned.
  - For QUESTION mode, this variable is set to the request details entered when the request for more information was submitted.
  - For ANSWER mode, this variable is set to the answering information provided in response to the request for more information.
- WF_ENGINE.context_recipient_role - The role currently designated as the recipient of the notification. This value may be the same as the value of the WF_ENGINE.context_user variable, or it may be a group role of which the context user is a member.
- WF_ENGINE.context_original_recipient - The role that has ownership of and responsibility for the notification. This value may differ from the value of the WF_ENGINE.context_recipient_role variable if the notification has previously been reassigned.
- WF_ENGINE.context_from_role - The role currently specified as the From role for the notification. This variable may be null if no From role is specified.
  - For RESPOND mode, the From role may be null or may be set by special logic in the workflow process. See: #FROM_ROLE Attribute, *Oracle Workflow Developer's Guide*.
  - For FORWARD or TRANSFER mode, the From role is the role that reassigned the notification.
  - For QUESTION mode, the From role is the role that sent the request for more information.
  - For ANSWER mode, the From role is the role that sent the answering information.
- WF_ENGINE.context_new_role - The new role to which the action on the notification is directed.
  - For RESPOND mode, this variable is null.
  - For FORWARD or TRANSFER mode, this variable is set to the new recipient role to which the notification is being reassigned.
  - For QUESTION mode, this variable is set to the role to which the request for more information is being sent.

    - For ANSWER mode, this variable is set to the role that sent the request for more information and is receiving the answer.
- WF_ENGINE.context_more_info_role - The role to which the most recent previous request for more information was sent. This variable may be null if no such request has previously been submitted for this notification.
- WF_ENGINE.context_user_key - If the notification was sent as part of a workflow process, and a user key is set for this process instance, then WF_ENGINE.context_user_key is set to that user key. Otherwise, this variable is null.
- WF_ENGINE.context_proxy - For RESPOND, FORWARD, TRANSFER, QUESTION, or ANSWER mode, if the user who took that action was acting on behalf of another user through the worklist access feature, then the value of WF_ENGINE.context_proxy is the user name of the logged in user who took the action. Otherwise, this variable is null.

You can reference these global engine variables in your PL/SQL function.

> **Note:** For RUN mode and TIMEOUT mode, only the WF_ENGINE.context_nid and WF_ENGINE.context_user variables are set.

> **Note:** The WF_ENGINE.context_text variable from earlier versions of Oracle Workflow is replaced by the WF_ENGINE.context_user and WF_ENGINE.context_new_role variables. The current version of Oracle Workflow still recognizes the WF_ENGINE.context_text variable for backward compatibility, but moving forward, you should only use the new WF_ENGINE.context_user and WF_ENGINE.context_new_role variables where appropriate.

Then when the notification's state changes, a notification callback function executes the post-notification function in the mode that matches the notification's state: RESPOND, FORWARD, TRANSFER, QUESTION, or ANSWER.

When a recipient responds, the Workflow Engine initially runs the post-notification function in VALIDATE mode which allows you to validate the response values before accepting the response. Then the Workflow Engine runs the post-notification function in RESPOND mode to record the response. Finally, when the Notification System completes execution of the post-notification function in RESPOND mode, the Workflow Engine automatically runs the post-notification function again in RUN mode. In this mode, the post-notification function can perform additional processing such as vote tallying.

If a notification activity times out, the Workflow Engine runs the post-notification function for the activity in TIMEOUT mode. For a Voting activity, the TIMEOUT mode logic should identify how to tally the votes received up until the timeout.

When the post-notification function completes, the Workflow Engine erases the global engine variables.

As a final step, if the post-notification function is run in TRANSFER mode and Expand Roles is not checked for the notification activity, the Workflow Engine sets the assigned user for the notification to the new role name specified.

> **Important:** If the post-notification function returns ERROR:*<errcode>* as a result or raises an exception, the Workflow Engine aborts the operation. For example, if the post-notification function is executed

in FORWARD mode and it raises an exception because the role being forwarded to is invalid, an error is displayed to the user and the Forward operation is not executed. The notification recipient is then prompted again to take some type of action.

See: Notification Model, page 4-1.

Synchronous, Asynchronous, and Forced Synchronous Processes

A workflow process can be either synchronous or asynchronous. A synchronous process is a process that can be executed without interruption from start to finish. The Workflow Engine executes a process synchronously when the process includes activities that can be completed immediately, such as function activities that are not deferred to the background engine. The Workflow Engine does not return control to the calling application that initiated the workflow until it completes the process. With a synchronous process, you can immediately check for process results that were written to item attributes or directly to the database. However, the user must wait for the process to complete.

An asynchronous process is a process that the Workflow Engine cannot complete immediately because it contains activities that interrupt the flow. Examples of activities that force an asynchronous process include deferred activities, notifications with responses, blocking activities, and wait activities. Rather than waiting indefinitely when it encounters one of these activities, the Workflow Engine sets the audit tables appropriately and returns control to the calling application. The workflow process is left in an unfinished state until it is started again. The process can be restarted by the Notification System, such as when a user responds to a notification; by the background engine, such as when a deferred activity is executed; or by the Business Event System, such as when an event message is dequeued from an inbound queue and sent to the workflow process. With an asynchronous process, the user does not have to wait for the process to complete to continue using the application. However, the results of the process are not available until the process is completed at a later time.

In addition to regular synchronous and asynchronous processes, the Workflow Engine also supports a special class of synchronous processes called forced synchronous processes. A forced synchronous process completes in a single SQL session from start to finish and never inserts into or updates any database tables. As a result, the execution speed of a forced synchronous process is significantly faster than a typical synchronous process. The process results are available immediately upon completion. However, no audit trail is recorded.

There may be cases when your application requires a forced synchronous process to generate a specific result quickly when recording an audit trail is not a concern. For example, in Oracle Applications, several products require Account Generator workflows to generate a meaningful flexfield code derived from a series of concatenated segments pulled from various tables. The Account Generator workflows are forced synchronous processes that compute and pass back completed flexfield codes to the calling applications instantaneously.

To create a forced synchronous process, you need to set the item key of your process to #SYNCH or to wf_engine.eng_synch, which returns the #SYNCH constant, when you call the necessary WF_ENGINE APIs. Since a forced synchronous process never writes to the database, using a non-unique item key such as #SYNCH is not an issue. Your process definition, however, must adhere to the following set of restrictions:

- No notification activities are allowed.
- Limited blocking-type activities are allowed. A process can block and restart with a call to WF_ENGINE.CompleteActivity only if the blocking and restarting activities:
  - Occur in the same database session.
  - Contain no intervening calls to Oracle Workflow.
  - Contain no intervening commits.
- No error processes can be assigned to the process or the process's activities.
- Each function activity behaves as if On Revisit is set to `Loop`, and is run in non-cancelling mode, regardless of its actual On Revisit setting. Loops are allowed in the process.
- No Master/Detail coordination activities are allowed.
- No parallel flows are allowed in the process, as transitions from each activity must have a distinct result. This also means that no `<Any>` transitions are allowed since they cause parallel flows.
- None of the following Standard activities are allowed:
  - And
  - Block (restricted by the conditions stated in the Limited Blocking bullet point above.)
  - Defer Thread
  - Wait
  - Continue Flow/Wait for Flow
  - Role Resolution
  - Voting
  - Compare Execution Time
  - Notify
- No use of the background engine, that is, activities are never deferred.
- No data is ever written to the Oracle Workflow tables and as a result:
  - The process cannot be viewed from the Workflow Monitor.
  - No auditing is available for the process.
- Only the following WF_ENGINE API calls are allowed to be made, and in all cases, the item key supplied to these APIs must be specified as `#SYNCH` or `wf_engine.eng_synch`:
  - WF_ENGINE.CreateProcess
  - WF_ENGINE.StartProcess
  - WF_ENGINE.GetItemAttribute
  - WF_ENGINE.SetItemAttribute
  - WF_ENGINE.GetActivityAttribute

- WF_ENGINE.CompleteActivity (for the limited usage of blocking-type activities)

- WF_ENGINE API calls for any item besides the current synchronous item are not allowed.

  Important: If you encounter an error from a forced synchronous process, you should rerun the process with a unique item key in asynchronous mode and check the error stack using the Workflow Monitor or the script `wfstat.sql`. If the synchronous process completes successfully, the error you encountered in the forced synchronous process is probably due to a violation of one of the above listed restrictions. See: Wfstat.sql, *Oracle Workflow Administrator's Guide*.

  Note: The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

See: Synchronous, Asynchronous, and Forced Synchronous Workflows, *Oracle Workflow Administrator's Guide*.

Business Events

Events from the Business Event System are represented within workflow processes as event activities. An event activity can either raise, send, or receive a business event.

A Raise event activity raises an event to the Event Manager, triggering any subscriptions to that event. The Workflow Engine calls the WF_EVENT.Raise API to raise the event. See: Raise, page 5-21.

A Send event activity sends an event directly to a Business Event System agent without raising the event to the Event Manager. The Workflow Engine calls the WF_EVENT.Send API to send the event. See: Send, page 5-25.

A Receive event activity receives an event from the Event Manager into a workflow process, which then continues the thread of execution from that activity. The Workflow Engine can receive an event into an activity in an existing process instance that is waiting for the event, using the correlation ID in the event message to match the event with the process to which it belongs. The Workflow Engine can also receive an event into a Receive event activity that is marked as a Start activity to launch a new workflow process. The WF_ENGINE.Event API is used to receive an event into a workflow process. See: Event, page 2-54.

See also: Managing Business Events, *Oracle Workflow Developer's Guide* and Event Activities, *Oracle Workflow Developer's Guide*.

Workflow Engine APIs

The Workflow Engine APIs can be called by an application program or a workflow function in the runtime phase to communicate with the engine and to change the status of each of the activities. These APIs are defined in a PL/SQL package called WF_ENGINE.

Many of these Workflow Engine APIs also have corresponding Java methods that you can call from any Java program to integrate with Oracle Workflow. The following list indicates whether the Workflow Engine APIs are available as PL/SQL functions/procedures, as Java methods, or both.

Important: Java is case-sensitive and all Java method names begin with a lower case letter to follow Java naming conventions.

- Event - PL/SQL, page 2-54
- HandleError - PL/SQL and Java, page 2-55
- SetItemParent - PL/SQL and Java, page 2-57
- ItemStatus - PL/SQL and Java, page 2-59
- getProcessStatus - Java, page 2-60

Related Topics

Standard API for PL/SQL Procedures Called by Function Activities, *Oracle Workflow Developer's Guide*

CreateProcess

PL/SQL Syntax

```
procedure CreateProcess
  (itemtype in varchar2,
   itemkey in varchar2,
   process in varchar2 default '',
   user_key in varchar2 default null,
   owner_role in varchar2 default null);
```

Java Syntax

```
public static boolean createProcess
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String process)
```

Description

Creates a new runtime process for an application item.

For example, a Requisition item type may have a Requisition Approval Process as a top level process. When a particular requisition is created, an application calls CreateProcess to set up the information needed to start the defined process.

> **Caution:** Although you can make a call to *CreateProcess()* and *StartProcess()* from a database trigger to initiate a workflow process, you should avoid doing so in certain circumstances. For example, if a database entity has headers, lines and details, and you initiate a workflow process from an `AFTER INSERT` trigger at the header-level of that entity, your workflow process may fail because some subsequent activity in the process may require information from the entity's lines or details level that is not yet populated.
>
> **Important:** The Workflow Engine always issues a savepoint before executing each activity in a process so that it can rollback to the previous activity in case an error occurs. For environments such as database triggers or distributed transactions that do not allow savepoints, the

Workflow Engine automatically traps "Savepoint not allowed" errors and defers the execution of the activity. If you initiate a workflow process from a database trigger, the Workflow Engine immediately defers the initial start activities to a background engine, so that they are no longer executing from a database trigger.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type. Item types are defined in the Workflow Builder.

itemkey
A string derived usually from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the new process and must be passed to all subsequent API calls for that process.

Note: The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

Note: You can pass `#SYNCH` as the itemkey to create a forced synchronous process. See: Synchronous, Asynchronous, and Forced Synchronous Processes, page 2-12.

process
An optional argument that allows the selection of a particular process for that item. Provide the process internal name. If process is null, the item type's selector function is used to determine the top level process to run. If you do not specify a selector function and this argument is null, an error will be raised.

user_key
A user-friendly key to assign to the item identified by the specified item type and item key. This argument is optional.

owner_role
A valid role to set as the owner of the item. This argument is optional.

Example

Example

The following code excerpt shows an example of how to call *createProcess()* in a Java program. The example code is from the `WFTest.java` program.

```
// create an item
if (WFEngineAPI.createProcess(ctx, iType, iKey, pr))
  System.out.println("Created Item");
  else
{
  Systm.out.println("createProcess failed");
  WFEngineAPI.showError(ctx);
}
```

SetItemUserKey

PL/SQL Syntax

```
procedure SetItemUserKey
  (itemtype in varchar2,
   itemkey in varchar2,
   userkey in varchar2);
```

Java Syntax

```
public static boolean setItemUserKey
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String userKey)
```

Description

Lets you set a user-friendly identifier for an item in a process, which is initially identified by an item type and item key. The user key is intended to be a user-friendly identifier to locate items in the Workflow Monitor and other user interface components of Oracle Workflow.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype or itemType
A valid item type.

itemkey or itemKey
A string generated usually from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

userkey or userKey
The user key to assign to the item identified by the specified item type and item key.

GetItemUserKey

PL/SQL Syntax

```
function GetItemUserKey
  (itemtype in varchar2,
   itemkey in varchar2)
  return varchar2;
```

Java Syntax

```
public static String getItemUserKey
  (WFContext wCtx,
   String itemType,
   String itemKey)
```

Description

Returns the user-friendly key assigned to an item in a process, identified by an item type and item key. The user key is a user-friendly identifier to locate items in the Workflow Monitor and other user interface components of Oracle Workflow.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype or itemType
A valid item type.

itemkey or itemKey
A string generated usually from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

GetActivityLabel

PL/SQL Syntax

```
function GetActivityLabel
  (actid in number)
  return varchar2;
```

Description

Returns the instance label of an activity, given the internal activity instance ID. The label returned has the following format, which is suitable for passing to other Workflow Engine APIs, such as CompleteActivity and HandleError, that accept activity labels as arguments:

<process_name>:*<instance_label>*

Arguments (input)

actid
An activity instance ID.

SetItemOwner

PL/SQL Syntax

```
procedure SetItemOwner
  (itemtype in varchar2,
   itemkey in varchar2,
   owner in varchar2);
```

Java Syntax

```
public static boolean setItemOwner
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String owner)
```

Description

A procedure to set the owner of existing items. The owner must be a valid role. Typically, the role that initiates a transaction is assigned as the process owner, so that any participant in that role can find and view the status of that process instance in the Workflow Monitor.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type. Item types are defined in the Workflow Builder.

itemkey
A string derived from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the new process and must be passed to all subsequent API calls for that process.

owner
A valid role.

Example

Example

The following code excerpt shows an example of how to call *setItemOwner()* in a Java program. The example code is from the `WFTest.java` program.

```
// set item owner
if (WFEngineAPI.setItemOwner(ctx, iType, iKey, owner))
  System.out.println("Set Item Owner: "+owner);
else
{
  System.out.println("Cannot set owner.");
  WFEngineAPI.showError(ctx);
}
```

StartProcess

PL/SQL Syntax

```
procedure StartProcess
  (itemtype in varchar2,
   itemkey in varchar2);
```

Java Syntax

```
public static boolean startProcess
  (WFContext wCtx,
   String itemType,
   String itemKey)
```

Description

Begins execution of the specified process. The engine locates the activity marked as `START` and then executes it. *CreateProcess()* must first be called to define the item type and item key before calling *StartProcess()*.

> **Caution:** Although you can make a call to *CreateProcess()* and *StartProcess()* from a trigger to initiate a workflow process, you should avoid doing so in certain circumstances. For example, if a database entity has headers, lines and details, and you initiate a workflow process from an `AFTER INSERT` trigger at the header-level of that entity, your workflow process may fail because some subsequent activity in the process may require information from the entity's lines or details level that is not yet populated.

> **Caution:** The Workflow Engine always issues a savepoint before executing each activity so that it can rollback to the previous activity in case an error occurs. Because of this feature, you should avoid initiating a workflow process from a database trigger because savepoints and rollbacks are not allowed in a database trigger.
>
> If you must initiate a workflow process from a database trigger, you must immediately defer the initial start activities to a background engine, so that they are no longer executing from a database trigger. To accomplish this:
>
> - Set the cost of the process start activities to a value greater than the Workflow Engine threshold (default value is 0.5)
>
>   or
>
> - Set the Workflow Engine threshold to be less than 0 before initiating the process:

```
begin
  save_threshold := WF_ENGINE.threshold;
  WF_ENGINE.threshold := -1;
  WF_ENGINE.CreateProcess(...);
  WF_ENGINE.StartProcess(...);
--Always reset threshold or all activities in this
--session will be deferred.
  WF_ENGINE.threshold := save_threshold;
end
```

(This method has the same effect as the previous method, but is more secure as the initial start activities are always deferred even if the activities' costs change.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string derived from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

> **Note:** You can pass `#SYNCH` as the item key to create a forced synchronous process. See: Synchronous, Asynchronous, and Forced Synchronous Processes, page 2-12.

> **Note:** The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

Example

Example

The following code excerpt shows an example of how to call *startProcess()* in a Java program. The example code is from the `WFTest.java` program.

```
// start a process
if (WFEngineAPI.startProcess(ctx, iType, iKey))
  System.out.println("Process Started successfully");
  else
{
  System.out.println("launch failed");
  WFEngineAPI.showError(ctx);
}
```

LaunchProcess

PL/SQL Syntax

```
procedure LaunchProcess
  (itemtype in varchar2,
   itemkey in varchar2,
   process in varchar2 default '',
   userkey in varchar2 default '',
   owner in varchar2 default '');
```

Java Syntax

```
public static boolean launchProcess
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String process,
   String userKey,
   String owner)
```

Description

Launches a specified process by creating the new runtime process and beginning its execution. This is a wrapper that combines CreateProcess and StartProcess.

Caution: Although you can make a call to *CreateProcess()* and *StartProcess()* from a database trigger to initiate a workflow process, you should avoid doing so in certain circumstances. For example, if a database entity has headers, lines and details, and you initiate a workflow process from an `AFTER INSERT` trigger at the header-level of that entity, your workflow process may fail because some subsequent activity in the process may require information from the entity's lines or details level that is not yet populated.

Important: The Workflow Engine always issues a savepoint before executing each activity in a process so that it can rollback to the previous activity in case an error occurs. For environments such as database triggers or distributed transactions that do not allow savepoints, the Workflow Engine automatically traps "Savepoint not allowed" errors and defers the execution of the activity. If you initiate a workflow process from a database trigger, the Workflow Engine immediately defers the initial start activities to a background engine, so that they are no longer executing from a database trigger.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string derived from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the new process and must be passed to all subsequent API calls for that process.

> **Note:** The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.
>
> You can pass `#SYNCH` as the item key to create a forced synchronous process. See: Synchronous, Asynchronous, and Forced Synchronous Processes, page 2-12.

process
An optional argument that allows the selection of a particular process for that item. Provide the process internal name. If process is null, the item type's selector function is used to determine the top level process to run. This argument defaults to null.

userkey
The user key to assign to the item identified by the specified item type and item key. If `userkey` is null, then no user key is assigned to the item instance.

owner
A valid role designated as the owner of the item. If `owner` is null, then no owner is assigned to the process and only the workflow administrator role can monitor the process.

SuspendProcess

PL/SQL Syntax

```
procedure SuspendProcess
  (itemtype in varchar2,
   itemkey in varchar2,
   process in varchar2 default '');
```

Java Syntax

```
public static boolean suspendProcess
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String process)
```

Description

Suspends process execution so that no new transitions occur. Outstanding notifications can complete by calling *CompleteActivity()*, but the workflow does not transition to the next activity. Restart suspended processes by calling *ResumeProcess()*.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

process
An optional argument that allows the selection of a particular subprocess for that item. Provide the process activity's label name. If the process activity label name does not uniquely identify the subprocess you can precede the label name with the internal name of its parent process. For example:

<parent_process_internal_name>:<label_name>
If this argument is null, the top level process for the item is suspended. This argument defaults to null.

Example

Example
The following code excerpt shows an example of how to call *suspendProcess()* in a Java program. The example code is from the `WFTest.java` program.

```
// suspend, status should become SUSPEND
System.out.println("Suspend Process " + iType +"/"+ iKey +
    " ...");
if (WFEngineAPI.suspendProcess(ctx, iType, iKey, null))
  System.out.println("Seems to suspend successfully");
else
{
  System.out.println("suspend failed");
  WFEngineAPI.showError(ctx);
}
```

ResumeProcess

PL/SQL Syntax

```
procedure ResumeProcess
  (itemtype in varchar2,
   itemkey in varchar2,
   process in varchar2 default '');
```

Java Syntax

```
public static boolean resumeProcess
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String process)
```

Description

Returns a suspended process to normal execution status. Any activities that were transitioned to while the process was suspended are now executed.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

process
An optional argument that allows the selection of a particular subprocess for that item type. Provide the process activity's label name. If the process activity label name does not uniquely identify the subprocess you can precede the label name with the internal name of its parent process. For example:

<parent_process_internal_name>:*<label_name>*

If this argument is null, the top level process for the item is resumed. This argument defaults to null.

Example

Example
The following code excerpt shows an example of how to call *resumeProcess()* in a Java program. The example code is from the `WFTest.java` program.

```
// resume process and status should be ACTIVE
System.out.println("Resume Process " + iType +"/"+ iKey +
    " ...");
if (WFEngineAPI.resumeProcess(ctx, iType, iKey, null))
  System.out.println("Seems to resume successfully");
else
{
  System.out.println("resume failed");
  WFEngineAPI.showError(ctx);
}
```

AbortProcess

PL/SQL Syntax

```
procedure AbortProcess
  (itemtype in varchar2,
   itemkey in varchar2,
   process in varchar2 default '',
   result in varchar2 default eng_force);
```

Java Syntax

```
public static boolean abortProcess
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String process,
   String result)
```

Description

Aborts process execution and cancels outstanding notifications. The process status is considered `COMPLETE`, with a result specified by the `result` argument. Also, any outstanding notifications or subprocesses are set to a status of `COMPLETE` with a result of `force`, regardless of the `result` argument.

This API also raises the `oracle.apps.wf.engine.abort` event. Although Oracle Workflow does not include any predefined subscriptions to this event, you can optionally define your own subscriptions to this event if you want to perform custom processing when it occurs. See: Workflow Engine Events, *Oracle Workflow Developer's Guide* and To Define an Event Subscription (for standalone Oracle Workflow), *Oracle Workflow Developer's Guide* or To Create or Update an Event Subscription (for Oracle Applications), *Oracle Workflow Developer's Guide*.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

process
An optional argument that allows the selection of a particular subprocess for that item type. Provide the process activity's label name. If the process activity label name does not uniquely identify the subprocess you can precede the label name with the internal name of its parent process. For example:

<parent_process_internal_name>:*<label_name>*
If this argument is null, the top level process for the item is aborted. This argument defaults to null.

result
A status assigned to the aborted process. The result must be one of the values defined in the process Result Type, or one of the following standard engine values:

- `eng_exception`
- `eng_timeout`
- `eng_force`
- `eng_mail`

- eng_null

This argument defaults to "eng_force".

Example

Example

The following code excerpt shows an example of how to call *abortProcess()* in a Java program. The example code is from the WFTest.java program.

```
// abort process, should see status COMPLETE with result
// code force
System.out.println("Abort Process ..." + iType + "/" +
    iKey);
if (!WFEngineAPI.abortProcess(ctx, iType, iKey, pr, null))
{
  System.out.println("Seemed to have problem aborting...");
  WFEngineAPI.showError(ctx);
}
```

CreateForkProcess

PL/SQL Syntax

```
procedure CreateForkProcess
  (copy_itemtype in varchar2,
   copy_itemkey in varchar2,
   new_itemkey in varchar2,
   same_version in boolean default TRUE);
```

Description

Forks a runtime process by creating a new process that is a copy of the original. After calling *CreateForkProcess()*, you can call APIs such as *SetItemOwner()*, *SetItemUserKey()*, or the *SetItemAttribute* APIs to reset any item properties or modify any item attributes that you want for the new process. Then you must call *StartForkProcess()* to start the new process.

Use *CreateForkProcess()* when you need to change item specific attributes during the course of a process. For example, if an order cannot be met due to insufficient inventory stock, you can use *CreateForkProcess()* to fork a new transaction for the backorder quantity. Note that any approval notification will be copied. The result is as if two items were created for this transaction.

Caution: Do not call *CreateForkProcess()* and *StartForkProcess()* from within a parallel branch in a process. These APIs do not copy any branches parallel to their own branch that are not active.

Note: When you fork an item, Oracle Workflow automatically creates an item attribute called #FORKED_FROM for the new item and sets the attribute to the item key of the original item. This attribute provides an audit trail for the forked item.

Arguments (input)

copy_itemtype
A valid item type for the original process to be copied. The new process will have the same item type.

copy_itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The copy item type and key together identify the original process to be copied.

new_itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and new item key together identify the new process.

> **Note:** The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

same_version
Specify `TRUE` or `FALSE` to indicate whether the new runtime process uses the same version as the original or the latest version. If you specify `TRUE`, *CreateForkProcess()* copies the item attributes and status of the original process to the new process. If you specify `FALSE`, *CreateForkProcess()* copies the item attributes of the original process to the new process but does not copy the status. Defaults to `TRUE`.

StartForkProcess

PL/SQL Syntax

```
procedure StartForkProcess
  (itemtype in varchar2,
   itemkey in varchar2);
```

Description

Begins execution of the new forked process that you specify. Before you call *StartForkProcess()*, you must first call *CreateForkProcess()* to create the new process. You can modify the item attributes of the new process before calling *StartForkProcess()*.

If the new process uses the same version as the original, *StartForkProcess()* copies the status and history of each activity in the forked process, activity by activity. If the new process uses the latest version, then *StartForkProcess()* executes *StartProcess()*.

If you call *StartForkProcess()* from within a process, any function activity in the process that had a status of '`Active`' is updated to have a status of '`Notified`'. You must call *CompleteActivity()* afterwards to continue the process.

StartForkProcess() automatically refreshes any notification attributes that are based on item attributes. Any open notifications in the original process are copied and sent again in the new process. Closed notifications are copied but not resent; their status remains remains '`Complete`'.

Any Wait activities in the new process are activated at the same time as the original activities. For example, if a 24 hour Wait activity in the original process is due to be eligible in two hours, the new Wait activity is also eligible in two hours.

Caution: Do not call *CreateForkProcess()* and *StartForkProcess()* from within a parallel branch in a process. These APIs do not copy any branches parallel to their own branch that are not active.

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process.

Note: The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

Background

PL/SQL Syntax

```
procedure Background
  (itemtype in varchar2,
   minthreshold in number default null,
   maxthreshold in number default null,
   process_deferred in boolean default TRUE,
   process_timeout in boolean default FALSE,
   process_stuck in boolean default FALSE);
```

Description

Runs a background engine for processing deferred activities, timed out activities, and stuck processes using the parameters specified. The background engine executes all activities that satisfy the given arguments at the time that the background engine is invoked. This procedure does not remain running long term, so you must restart this procedure periodically. Any activities that are newly deferred or timed out or processes that become stuck after the current background engine starts are processed by the next background engine that is invoked. You can run a script called `wfbkgchk.sql` to get a list of the activities waiting to be processed by the next background engine run. See: Wfbkgchk.sql, *Oracle Workflow Administrator's Guide*.

You must not call *Background()* from within application code. If you want to call this procedure directly, you can run it from SQL*Plus. Otherwise, if you are using the standalone version of Oracle Workflow, you can use one of the sample background engine looping scripts described below, create your own script to make the background engine procedure loop indefinitely, or use the Oracle Workflow Manager component of Oracle Enterprise Manager to schedule a background engine. If you are using the version of Oracle Workflow embedded in Oracle Applications, you can use the concurrent program version of this procedure and take advantage of the concurrent manager to schedule the background engine to run periodically. You can also use the Workflow Manager component of Oracle Applications Manager to submit the background engine concurrent program. See: To Schedule Background Engines, *Oracle Workflow Administrator's Guide*.

Arguments (input)

itemtype
A valid item type. If the item type is null the Workflow engine will run for all item types.

minthreshold
Optional minimum cost threshold for an activity that this background engine processes, in hundredths of a second. There is no minimum cost threshold if this parameter is null.

maxthreshold
Optional maximum cost threshold for an activity that this background engine processes in hundredths of a second. There is no maximum cost threshold if this parameter is null.

process_deferred
Specify `TRUE` or `FALSE` to indicate whether to run deferred processes. Defaults to `TRUE`.

process_timeout
Specify `TRUE` or `FALSE` to indicate whether to run timed out processes. Defaults to `FALSE`.

process_stuck
Specify `TRUE` or `FALSE` to indicate whether to run stuck processes. Defaults to `FALSE`.

Example Background Engine Looping Scripts

Example

For the standalone version of Oracle Workflow you can use one of two example scripts to run the background engine regularly.

The first example is a SQL script stored in a file called `wfbkg.sql` in the *ORACLE_HOME*`/wf/admin/sql` directory. To run this script, go to the directory where the file is located and type the following command at your operating system prompt:

```
sqlplus <username/password> @wfbkg <min> <sec>
```

Replace *<username/password>* with the Oracle Database account username and password where you want to run the background engine. Replace *<min>* with the number of minutes you want the background engine to run and replace *<sec>* with the number of seconds you want the background engine to sleep between calls.

The second example is a shell script stored in a file called `wfbkg.csh` in the *ORACLE_HOME*`/bin` directory. To run this script, go to the directory where the file is located and type the following command at your operating system prompt:

```
wfbkg.csh <username/password>
```

Replace *<username/password>* with the Oracle Database account username and password where you want to run the background engine.

AddItemAttribute

PL/SQL Syntax

```
procedure AddItemAttr
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in varchar2,
   text_value in varchar2 default null,
   number_value in number default null,
   date_value in date default null);
```

Java Syntax

```
public static boolean addItemAttr
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String aName)

public static boolean addItemAttrText
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String aName,
   String aValue)

public static boolean addItemAttrNumber
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String aName,
   BigDecimal numberVal)

public static boolean addItemAttrDate
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String aName,
   String aValue)
```

Description

Adds a new item type attribute variable to the process. Although most item type attributes are defined at design time, you can create new attributes at runtime for a specific process. You can optionally set a default text, number, or date value for a new item type attribute when the attribute is created.

If you are using Java, choose the correct method for your attribute type. To add an empty item type attribute, use *addItemAttr()*. When adding an item type attribute with a default value, use *addItemAttrText()* for all attribute types except number and date.

> **Note:** If you need to add large numbers of item type attributes at once, use the *AddItemAttributeArray* APIs rather than the *AddItemAttribute* APIs for improved performance. See: AddItemAttributeArray, page 2-34

Arguments (input)

wCtx
Workflow context information. Required for the Java methods only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

aname
The internal name of the item type attribute.

text_value
The default text value for the item type attribute. Required for the PL/SQL procedure only. Defaults to null.

number_value or numberVal
The default number value for the item type attribute. Required for the PL/SQL procedure and *addItemAttrNumber()* Java method only. Defaults to null.

date_value
The default date value for the item type attribute. Required for the PL/SQL procedure only. Defaults to null.

aValue
The default value for the item type attribute. Required for the *addItemAttrText()* and *addItemAttrDate()* Java methods only.

Example

Example

The following example shows how API calls can be simplified by using *AddItemAttr()* to set the default value of a new item type attribute at the time of creation.

Using *AddItemAttr()* to create the new attribute and *SetItemAttrText()* to set the value of the attribute, the following calls are required:

```
AddItemAttr('ITYPE', 'IKEY', 'NEWCHAR_VAR');
SetItemAttrText('ITYPE', 'IKEY', 'NEWCHAR_VAR',
                'new text values');
```

Using *AddItemAttr()* both to create the new attribute and to set its value, only the following call is required:

```
AddItemAttr('ITYPE', 'IKEY', 'NEWCHAR_VAR',
            'new text values');
```

AddItemAttributeArray

PL/SQL Syntax

```
procedure AddItemAttrTextArray
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in Wf_Engine.NameTabTyp,
   avalue in Wf_Engine.TextTabTyp);

procedure AddItemAttrNumberArray
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in Wf_Engine.NameTabTyp,
   avalue in Wf_Engine.NumTabTyp);

procedure AddItemAttrDateArray
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in Wf_Engine.NameTabTyp,
   avalue in Wf_Engine.DateTabTyp);
```

Description

Adds an array of new item type attributes to the process. Although most item type attributes are defined at design time, you can create new attributes at runtime for a specific process. Use the *AddItemAttributeArray* APIs rather than the *AddItemAttribute* APIs for improved performance when you need to add large numbers of item type attributes at once.

Use the correct procedure for your attribute type. All attribute types except number and date use *AddItemAttrTextArray.*

> **Note:** The *AddItemAttributeArray* APIs use PL/SQL table composite datatypes defined in the WF_ENGINE package. The following table shows the column datatype definition for each PL/SQL table type.

PL/SQL Table Types in WF_ENGINE

PL/SQL Table Type	Column Datatype Definition
NameTabTyp	Wf_Item_Attribute_Values. NAME%TYPE
TextTabTyp	Wf_Item_Attribute_Values. TEXT_VALUE%TYPE
NumTabTyp	Wf_Item_Attribute_Values. NUMBER_VALUE%TYPE
DateTabTyp	Wf_Item_Attribute_Values. DATE_VALUE%TYPE

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

aname
An array of the internal names of the new item type attributes.

avalue
An array of the values for the new item type attributes.

SetItemAttribute

PL/SQL Syntax

```
procedure SetItemAttrText
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in varchar2,
   avalue in varchar2);

procedure SetItemAttrNumber
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in varchar2,
   avalue in number);

procedure SetItemAttrDate
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in varchar2,
   avalue in date);

procedure SetItemAttrEvent
  (itemtype in varchar2,
   itemkey in varchar2,
   name in varchar2,
   event in wf_event_t);
```

Java Syntax

```
public static boolean setItemAttrText
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String aName,
   String aValue)

public static boolean setItemAttrNumber
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String aName,
   BigDecimal aValue)

public static boolean setItemAttrDate
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String aName,
   String aValue)

public static boolean setItemAttrDate
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String attributeName,
   java.util.Date attributeValue)
```

Description

Sets the value of an item type attribute in a process. Use the correct procedure for your attribute type. All attribute types except number, date, and event use *SetItemAttrText*.

In Java, there are two implementations of *setItemAttrDate()*. One lets you provide the date value as a Java String object, while the other lets you provide the date value as a Java Date object.

> **Note:** If you need to set the values of large numbers of item type attributes at once, use the *SetItemAttributeArray* APIs rather than the *SetItemAttribute* APIs for improved performance. See: Set ItemAttributeArray, page 2-39.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

aname, name, or attributeName
The internal name of the item type attribute.

avalue, event, or attributeValue
The value for the item type attribute.

Examples

Example 1

The following code excerpt shows an example of how to call *setItemAttrText()* in a Java program. The example code is from the `WFTest.java` program.

```
if (WFEngineAPI.setItemAttrText(ctx, iType, iKey,
    "REQUESTOR_USERNAME", owner))
  System.out.println("Requestor: "+owner);
  else
{
  WFEngineAPI.showError(ctx);
}
```

Example 2

If an event message is stored within an item attribute of type event, you can access the event data CLOB within that event message by creating an item attribute of type URL for the event data. The following sample PL/SQL code shows how to set the value of the URL attribute in the standalone version of Oracle Workflow to reference the event data.

```
l_eventdataurl := Wfa_html.base_url||'Wf_Event_Html.
EventDataContents?P_EventAttribute=EVENT_MESSAGE'||'&'||
'P_ItemType='||itemtype||'&'||'P_ItemKey='||itemkey||'&'||
'p_mime_type=text/xml';

WF_ENGINE.SetItemAttrText('<item_type>', '<item_key>',
          'EVENTDATAURL', l_eventdataurl);
```

If you have applied a stylesheet to the event data XML document to create HTML, set the `p_mime_type` parameter in the URL to `text/html` instead.

If you omit the `p_mime_type` parameter from the URL, the MIME type defaults to `text/xml`.

Related Topics

Event Message Structure, page 5-6

setItemAttrFormattedDate

Java Syntax

```
public static boolean setItemAttrFormattedDate
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String attributeName,
   String attributeValue
   String dateFormat)
```

Description

Sets the value of an item type attribute of type date in a process with a date value provided as a formatted string.

Arguments (input)

wCtx
Workflow context information. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

attributeName
The internal name of the item type attribute.

attributeValue
The date value for the item type attribute.

dateFormat
The format of the date value. The format must be a date format mask that is supported by the Oracle Database. If no format is provided, the default value is the canonical date format for the database. See: Date Formats, *Oracle Database Globalization Support Guide*.

SetItemAttrDocument

Important: Document management functionality is reserved for future use. This description of the SetItemAttrDocument API is provided for reference only.

PL/SQL Syntax

```
procedure SetItemAttrDocument
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in varchar2,
   documentid in varchar2);
```

Java Syntax

```
public static boolean setItemAttrDocument
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String aName,
   String documentId)
```

Description

Sets the value of an item attribute of type document, to a document identifier.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

aname
The internal name of the item type attribute.

documentid
The value for the item type attribute as a fully concatenated string of the following values:

`DM:<node_id>:<doc_id>:<version>`

- *<node_id>* is the node ID assigned to the document management system node as defined in the Document Management Nodes Web page.
- *<doc_id>* is the document ID of the document, as assigned by the document management system where the document resides.
- *<version>* is the version of the document. If a version is not specified, the latest version is assumed.

SetItemAttributeArray

PL/SQL Syntax

```
procedure SetItemAttrTextArray
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in Wf_Engine.NameTabTyp,
   avalue in Wf_Engine.TextTabTyp);

procedure SetItemAttrNumberArray
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in Wf_Engine.NameTabTyp,
   avalue in Wf_Engine.NumTabTyp);

procedure SetItemAttrDateArray
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in Wf_Engine.NameTabTyp,
   avalue in Wf_Engine.DateTabTyp);
```

Description

Sets the values of an array of item type attributes in a process. Use the *SetItemAttributeArray* APIs rather than the *SetItemAttribute* APIs for improved performance when you need to set the values of large numbers of item type attributes at once.

Use the correct procedure for your attribute type. All attribute types except number, date, and event use *SetItemAttrTextArray*.

> **Note:** The *SetItemAttributeArray* APIs use PL/SQL table composite datatypes defined in the WF_ENGINE package. The following table shows the column datatype definition for each PL/SQL table type.

PL/SQL Table Types in WF_ENGINE

PL/SQL Table Type	Column Datatype Definition
NameTabTyp	Wf_Item_Attribute_Values. NAME%TYPE
TextTabTyp	Wf_Item_Attribute_Values. TEXT_VALUE%TYPE
NumTabTyp	Wf_Item_Attribute_Values. NUMBER_VALUE%TYPE
DateTabTyp	Wf_Item_Attribute_Values. DATE_VALUE%TYPE

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

aname
An array of the internal names of the item type attributes.

avalue
An array of the values for the item type attributes.

Example

Example

The following example shows how using the *SetItemAttributeArray* APIs rather than the *SetItemAttribute* APIs can help reduce the number of calls to the database.

Using *SetItemAttrText()*:

```
SetItemAttrText('ITYPE', 'IKEY', 'VAR1', 'value1');
SetItemAttrText('ITYPE', 'IKEY', 'VAR2', 'value2');
SetItemAttrText('ITYPE', 'IKEY', 'VAR3', 'value3');

// Multiple calls to update the database.
```

Using *SetItemAttrTextArray()*:

```
declare
  varname   Wf_Engine.NameTabTyp;
  varval    Wf_Engine.TextTabTyp;
begin
  varname(1) := 'VAR1';
  varval(1)  := 'value1';
  varname(2) := 'VAR2';
  varval(2)  := 'value2';
  varname(3) := 'VAR3';
  varval(3)  := 'value3';
Wf_Engine.SetItemAttrTextArray('ITYPE', 'IKEY', varname, varval);
exception
  when OTHERS then
    // handle your errors here
    raise;
end;

// Only one call to update the database.
```

getItemTypes

Java Syntax

```
public static WFTwoDArray getItemTypes
  (WFContext wCtx)
```

Description

Returns a list of all the item types defined in the Oracle Workflow database as a two-dimensional data object.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

GetItemAttribute

PL/SQL Syntax

```
function GetItemAttrText
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in varchar2,
   ignore_notfound in boolean default FALSE)
  return varchar2;

function GetItemAttrNumber
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in varchar2,
   ignore_notfound in boolean default FALSE)
  return number;

function GetItemAttrDate
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in varchar2,
   ignore_notfound in boolean default FALSE)
  return date;

function GetItemAttrEvent
  (itemtype in varchar2,
   itemkey in varchar2,
   name in varchar2)
  return wf_event_t;
```

Description

Returns the value of an item type attribute in a process. Use the correct function for your attribute type. All attribute types except number, date, and event use *GetItemAttrText*.

For *GetItemAttrText()*, *GetItemAttrNumber()*, and *GetItemAttrDate()*, you can specify `TRUE` for the `ignore_notfound` parameter to ignore the exception encountered if the specified item type attribute does not exist. In this case the function returns a null value but does not raise an exception. For example, you can use this parameter if a new item type attribute is added to an item type, and your code needs to handle both the earlier version and the upgraded version of the item type.

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

aname
The internal name of an item type attribute, for *GetItemAttrText()*, *GetItemAttrNumber()*, and *GetItemAttrDate()*.

name
The internal name of an item type attribute, for *GetItemAttrEvent()*.

ignore_notfound
Specify `TRUE` or `FALSE` to indicate whether to ignore the exception if the specified item type attribute does not exist, for *GetItemAttrText()*, *GetItemAttrNumber()*, and *GetItemAttrDate()*. If you specify `TRUE` and the item type attribute you specify does not exist, the function returns a null value but does not raise an exception. Defaults to `FALSE`.

Related Topics

Event Message Structure, page 5-6

GetItemAttrDocument

Important: Document management functionality is reserved for future use. This description of the GetItemAttrDocument API is provided for reference only.

PL/SQL Syntax

```
function GetItemAttrDocument
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in varchar2,
   ignore_notfound in boolean default FALSE)
  return varchar2;
```

Description

Returns the document identifier for a DM document-type item attribute. The document identifier is a concatenated string of the following values:

`DM:`*`<nodeid>`*`:`*`<documentid>`*`:`*`<version>`*

`<nodeid>` is the node ID assigned to the document management system node as defined in the Document Management Nodes Web page.

`<documentid>` is the document ID of the document, as assigned by the document management system where the document resides.

`<version>` is the version of the document. If a version is not specified, the latest version is assumed.

You can specify `TRUE` for the `ignore_notfound` parameter to ignore the exception encountered if the specified item type attribute does not exist. In this case the function returns a null value but does not raise an exception. For example, you can use this parameter if a new item type attribute is added to an item type, and your code needs to handle both the earlier version and the upgraded version of the item type.

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

aname
The internal name of the item type attribute.

ignore_notfound
Specify `TRUE` or `FALSE` to indicate whether to ignore the exception if the specified item type attribute does not exist. If you specify `TRUE` and the item type attribute you specify does not exist, the function returns a null value but does not raise an exception. Defaults to `FALSE`.

GetItemAttrClob

PL/SQL Syntax

```
function GetItemAttrClob
  (itemtype in varchar2,
   itemkey in varchar2,
   aname in varchar2)
  return clob;
```

Description

Returns the value of an item type attribute in a process as a character large object (CLOB).

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

aname
The internal name of an item type attribute.

getItemAttributes

Java Syntax

```
public static WFTwoDArray getItemAttributes
  (WFContext wCtx,
   String itemType,
   String itemKey)
```

Description

Returns a list of all the item attributes, their types, and their values for the specified item type instance as a two-dimensional data object.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

GetItemAttrInfo

PL/SQL Syntax

```
procedure GetItemAttrInfo
  (itemtype in varchar2,
   aname in varchar2,
   atype out varchar2,
   subtype out varchar2,
   format out varchar2);
```

Description

Returns information about an item type attribute, such as its type and format, if any is specified. Currently, subtype information is not available for item type attributes.

Arguments (input)

itemtype
A valid item type.

aname
The internal name of an item type attribute.

GetActivityAttrInfo

PL/SQL Syntax

```
procedure GetActivityAttrInfo
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number,
   aname in varchar2,
   atype out varchar2,
   subtype out varchar2,
   format out varchar2);
```

Description

Returns information about an activity attribute, such as its type and format, if any is specified. This procedure currently does not return any subtype information for activity attributes.

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

actid
The activity ID for a particular usage of an activity in a process definition. Also referred to as the activity ID of the node.

aname
The internal name of an activity attribute.

GetActivityAttribute

PL/SQL Syntax

```
function GetActivityAttrText
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number,
   aname in varchar2,
   ignore_notfound in boolean default FALSE)
  return varchar2;

function GetActivityAttrNumber
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number,
   aname in varchar2,
   ignore_notfound in boolean default FALSE)
  return number;

function GetActivityAttrDate
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number,
   aname in varchar2,
   ignore_notfound in boolean default FALSE)
  return date;

function GetActivityAttrEvent
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number,
   name in varchar2)
  return wf_event_t;
```

Description

Returns the value of an activity attribute in a process. Use the correct function for your attribute type. If the attribute is a Number or Date type, then the appropriate function translates the number/date value to a text-string representation using the attribute format.

> **Note:** Use *GetActivityAttrText()* for form, URL, lookup, role, attribute, and document attribute types.

For *GetActivityAttrText()*, *GetActivityAttrNumber(),* and *GetActivityAttrDate()*, you can specify `TRUE` for the `ignore_notfound` parameter to ignore the exception encountered if the specified activity attribute does not exist. In this case the function returns a null value but does not raise an exception. For example, you can use this parameter if a new activity attribute is added to an activity, and your code needs to handle both the earlier version and the upgraded version of the activity.

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

actid
The activity ID for a particular usage of an activity in a process definition. Also referred to as the activity ID of the node.

aname
The internal name of an activity attribute, for *GetActivityAttrText()*, *GetActivityAttrNumber(),* and *GetActivityAttrDate()*.

name
The internal name of an activity attribute, for *GetActivityAttrEvent()*.

ignore_notfound
Specify `TRUE` or `FALSE` to indicate whether to ignore the exception if the specified activity attribute does not exist, for *GetActivityAttrText()*, *GetActivityAttrNumber(),* and *GetActivityAttrDate()*. If you specify `TRUE` and the activity attribute you specify does not exist, the function returns a null value but does not raise an exception. Defaults to `FALSE`.

Related Topics

Event Message Structure, page 5-6

GetActivityAttrClob

PL/SQL Syntax

```
function GetActivityAttrClob
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number,
   aname in varchar2)
  return clob;
```

Description

Returns the value of an activity attribute in a process as a character large object (CLOB).

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

actid
The activity ID for a particular usage of an activity in a process definition. Also referred to as the activity ID of the node.

aname
The internal name of an activity attribute.

getActivityAttributes

Java Syntax

```
public static WFTwoDArray getActivityAttributes
  (WFContext wCtx,
   String itemType,
   String itemKey,
   BigDecimal actID)
```

Description

Returns a list of all the activity attributes, their types, and their values for the specified activity as a two-dimensional data object.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

actID
The activity ID for a particular usage of an activity in a process definition. Also referred to as the activity ID of the node.

BeginActivity

PL/SQL Syntax

```
procedure BeginActivity
  (itemtype in varchar2,
   itemkey in varchar2,
   activity in varchar2);
```

Description

Determines if the specified activity can currently be performed on the process item and raises an exception if it cannot.

The *CompleteActivity()* procedure automatically performs this function as part of its validation. However, you can use *BeginActivity()* to verify that the activity you intend to perform is currently allowed before actually calling it. See: CompleteActivity, page 2-50.

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process.

activity
The activity node to perform on the process. Provide the activity node's label name. If the activity node label name does not uniquely identify the activity node you can precede the label name with the internal name of its parent process. For example:

<parent_process_internal_name>:*<label_name>*

Example

Example

```
/* Verify that a credit check can be performed on an order.  If i
t
 * is allowed, perform the credit check, then notify the Workflow
 * Engine when the credit check completes. */

begin
  wf_engine.BeginActivity('ORDER', to_char(order_id),
    'CREDIT_CHECK');
  OK := TRUE;
exception
  when others then
    WF_CORE.Clear;
    OK := FALSE;
end;

if OK then
  -- perform activity --
  wf_engine.CompleteActivity('ORDER', to_char(order_id),
    'CREDIT_CHECK' :result_code);
end if;
```

CompleteActivity

PL/SQL Syntax

```
procedure CompleteActivity
  (itemtype in varchar2,
   itemkey in varchar2,
   activity in varchar2,
  result in varchar2);
```

Java Syntax

```
public static boolean completeActivity
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String activity,
   String result)
```

Description

Notifies the Workflow Engine that the specified activity has been completed for a particular item. This procedure can be called for the following situations:

- **To indicate a completed activity with an optional result** - This signals the Workflow Engine that an asynchronous activity has been completed. This procedure requires that the activity currently has a status of 'Notified'. An optional activity completion result can also be passed. The result can determine what transition the process takes next.

- **To create and start an item** - You can call *CompleteActivity()* for a 'Start' activity to implicitly create and start a new item. 'Start' activities are designated as the beginning of a process in the Workflow Builder. The item type and key specified in this call must be passed to all subsequent calls that operate on this item.

  Use *CompleteActivity()* if you cannot use *CreateProcess()* and *StartProcess()* to start your process. For example, call *CompleteActivity()* if you need to start a process with an activity node that is mid-stream in a process thread and not at the beginning of a process thread. The activity node you specify as the beginning of the process must be set to '`Start`' in the Node tab of its property page or else an error will be raised.

  Note: Starting a process using *CompleteActivity()* differs from starting a process using *CreateProcess()* and *StartProcess()* in these ways:

  - The 'Start' activity called with *CompleteActivity()* may or may not have incoming transitions. *StartProcess()* executes only 'Start' activities that do not have any incoming transitions.
  - *CompleteActivity()* only completes the single 'Start' activity with which it is called. Other 'Start' activities in the process are not completed. *StartProcess()*, however, executes every activity in the process that is marked as a 'Start' activity and does not have any incoming transitions.
  - *CompleteActivity()* does not execute the activity with which it is called; it simply marks the activity as complete. *StartProcess()* does execute the 'Start' activities with which it starts a process.
  - When you use *CompleteActivity()* to start a new process, the item type of the activity being completed must either have a selector function defined to choose a root process, or have exactly one runnable process with the activity being completed marked as a 'Start' activity. You cannot explicitly specify a root process as you can with *StartProcess()*.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype or itemType
A valid item type.

itemkey or itemKey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process.

activity
The name of the activity node that is completed. Provide the activity node's label name. If the activity node label name does not uniquely identify the subprocess you can precede the label name with the internal name of its parent process. For example:

<parent_process_internal_name>:*<label_name>*

This activity node must be marked as a 'Start' activity.

result
An optional activity completion result. Possible values are determined by the process activity's Result Type, or one of the engine standard results. See: AbortProcess, page 2-26.

Examples

Example 1

```
/* Complete the 'ENTER ORDER' activity for the 'ORDER' item type.

 * The 'ENTER ORDER' activity allows creation of new items since
 * it is the start of a workflow, so the item is created by this
 * call as well. */

wf_engine.CompleteActivity('ORDER', to_char(order.order_id),
  'ENTER_ORDER', NULL);
```

Example 2

```
/* Complete the 'LEGAL REVIEW' activity with status 'APPROVED'.
 * The item must already exist. */

wf_engine.CompleteActivity('ORDER', '1003', 'LEGAL_REVIEW',
  'APPROVED');
```

Example 3

```
/* Complete the BLOCK activity which is used in multiple
 * subprocesses in parallel splits. */

wf_engine.CompleteActivity('ORDER', '1003',
  'ORDER_PROCESS:BLOCK-3', 'null');
```

CompleteActivityInternalName

PL/SQL Syntax

```
procedure CompleteActivityInternalName
  (itemtype in varchar2,
   itemkey in varchar2,
   activity in varchar2,
   result in varchar2);
```

Description

Notifies the Workflow Engine that the specified activity has been completed for a particular item. This procedure requires that the activity currently has a status of 'Notified'. An optional activity completion result can also be passed. The result can determine what transition the process takes next.

CompleteActivityInternalName() is similar to *CompleteActivity()* except that *CompleteActivityInternalName()* identifies the activity to be completed by the activity's internal name, while *CompleteActivity()* identifies the activity by the activity node label name. You should only use *CompleteActivityInternalName()* when you do not know the activity node label name. If you do know the activity node label name, use *CompleteActivity()* instead. See: CompleteActivity, page 2-50.

Note: Unlike *CompleteActivity()*, you cannot use *CompleteActivity InternalName()* to start a process. Also, you cannot use *CompleteActivityInternalName()* with a synchronous process.

When *CompleteActivityInternalName()* is executed, there must be exactly one instance of the specified activity with a status of 'Notified'. If there are multiple instances of the activity with 'Notified' statuses, the process enters an 'ERROR' state.

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process.

activity
The internal name of the activity that is completed. If the activity internal name does not uniquely identify the subprocess you can precede the activity internal name with the internal name of its parent process. For example:

<parent_process_internal_name>:*<activity_internal_name>*

result
An optional activity completion result. Possible values are determined by the process activity's result type, or one of the engine standard results. See: AbortProcess, page 2-26.

AssignActivity

PL/SQL Syntax

```
procedure AssignActivity
  (itemtype in varchar2,
   itemkey in varchar2,
   activity in varchar2,
   performer in varchar2);
```

Description

Assigns or reassigns an activity to another performer. This procedure may be called before the activity is transitioned to. For example, a function activity earlier in the process may determine the performer of a later activity.

If a new user is assigned to a notification activity that already has an outstanding notification, the outstanding notification is canceled and a new notification is generated for the new user by calling *WF_Notification.Transfer*.

Arguments (input)

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process.

activity
The label name of the activity node. If the activity node label name does not uniquely identify the activity node you can precede the label name with the internal name of its parent process. For example:

<parent_process_internal_name>:*<label_name>*

performer
The name of the user who will perform the activity (the user who receives the notification). The name should be a role name from the Oracle Workflow directory service.

Event

PL/SQL Syntax

```
procedure Event
  (itemtype in varchar2,
   itemkey in varchar2,
   process_name in varchar2 default null,
   event_message in wf_event_t);
```

Description

Receives an event from the Business Event System into a workflow process.

If the specified item key already exists, the event is received into that item. If the item key does not already exist, but the specified process includes an eligible Receive event activity marked as a Start activity, the Workflow Engine creates a new item running that process.

Within the workflow process that receives the event, the procedure searches for eligible Receive event activities. For an activity to be eligible to receive an event, its event filter must either be set to that particular event, set to an event group of which that event is a member, or left blank to accept any event. Additionally, the activity must either be marked as a Start activity, or it must have an activity status of `NOTIFIED`, meaning the process has transitioned to that activity and is waiting to receive the event.

For each eligible Receive event activity, *Event()* stores the event name, event key, and event message in the item type attributes specified in the event activity node, if they have been defined. Additionally, the procedure sets any parameters in the event message parameter list as item type attributes for the process, creating new item type attributes if a corresponding attribute does not already exist for any parameter. It also sets the subscription's globally unique identifier (GUID) as a dynamic item attribute so that the workflow process can reference other information in the subscription definition. Then the Workflow Engine begins a thread of execution from the event activity.

If no eligible Receive event activity exists for a received event, the procedure returns an exception and an error message.

> **Note:** If an event arrives at a Start activity to launch a new process instance, the Workflow Engine also searches for all other receive event activities that are marked as Start activities and that do not have

any incoming transitions, regardless of their event filter. For these activities, the Workflow Engine sets the activity status to NOTIFIED so that they will be ready to receive an event if any more events are sent to this process. This feature lets you design a workflow process that requires multiple events to be received when you do not know in advance the order in which the events will arrive.

Note: If the event received by a Receive event activity was originally raised by a Raise event activity in another workflow process, the item type and item key for that process are included in the parameter list within the event message. In this case, the Workflow Engine automatically sets the specified process as the parent for the process that receives the event, overriding any existing parent setting.

Arguments (input)

itemtype
A valid item type.

itemkey
A string that uniquely identifies the item within an item type. The item type and key together identify the process.

Note: The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

process_name
An optional argument that allows the selection of a particular subprocess for that item type. Provide the process activity's label name. If the process activity label name does not uniquely identify the subprocess you can precede the label name with the internal name of its parent process. For example:

<parent_process_internal_name>:*<label_name>*

If this argument is null, the top level process for the item is started. This argument defaults to null.

event_message
The event message containing the details of the event.

HandleError

PL/SQL Syntax

```
procedure HandleError
  (itemtype in varchar2,
   itemkey in varchar2,
   activity in varchar2,
   command in varchar2,
   result in varchar2);
```

Java Syntax

```
public static boolean handleError
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String activity,
   String command,
   String result)
```

Description

This procedure is generally called from an activity in an ERROR process to handle any process activity that has encountered an error.

You can also call this procedure for any arbitrary activity in a process, to roll back part of your process to that activity. The activity that you call this procedure with can have any status and does not need to have been executed. The activity can also be in a subprocess. If the activity node label is not unique within the process you can precede the activity node label name with the internal name of its parent process. For example:

<parent_process_internal_name>:*<label_name>*

This procedure clears the activity specified and all activities following it that have already been transitioned to by reexecuting each activity in CANCEL mode. For an activity in the 'Error' state, there are no other executed activities following it, so the procedure simply clears the errored activity.

Once the activities are cleared, this procedure resets any parent processes of the specified activity to a status of 'Active', if they are not already active.

The procedure then handles the specified activity based on the command you provide: SKIP or RETRY.

This API also raises the oracle.apps.wf.engine.skip event or the oracle.apps.wf.engine.retry event, depending on the command you provide. Although Oracle Workflow does not include any predefined subscriptions to these events, you can optionally define your own subscriptions to these events if you want to perform custom processing when they occur. See: Workflow Engine Events, *Oracle Workflow Developer's Guide* and To Define an Event Subscription (for standalone Oracle Workflow), *Oracle Workflow Developer's Guide* or To Create or Update an Event Subscription (for Oracle Applications), *Oracle Workflow Developer's Guide*.

> **Note:** An item's active date and the version number of the process that the item is transitioning through can never change once an item is created. Occasionally, however, you may want to use HandleError to manually make changes to your process for an existing item.
>
> If the changes you make to a process are minor, you can use HandleError to manually push an item through activities that will error or redirect the item to take different transitions in the process.
>
> If the changes you want to make to a process are extensive, then you need to perform at least the following steps:
>
> - Abort the process by calling *WF_ENGINE.AbortProcess()*.
> - Purge the existing item by calling *WF_PURGE.Items()*.
> - Revise the process.

- Recreate the item by calling *WF_ENGINE.CreateProcess()*.
- Restart the revised process at the appropriate activity by calling *WF_ENGINE.HandleError()*.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

item_type or itemType
A valid item type.

item_key or itemKey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process.

activity
The activity node that encountered the error or that you want to undo. Provide the label name of the activity node. If the activity node label name does not uniquely identify the subprocess you can precede the label name with the internal name of its parent process. For example:

<parent_process_internal_name>:*<label_name>*

command
One of two commands that determine how to handle the process activity:

- `SKIP` - do not reexecute the activity, but mark the activity as complete with the supplied result and continue execution of the process from that activity.
- `RETRY` - reexecute the activity and continue execution of the process from that activity.

result
The result you wish to supply if the command is `SKIP`.

SetItemParent

PL/SQL Syntax

```
procedure SetItemParent
  (itemtype in varchar2,
   itemkey in varchar2,
   parent_itemtype in varchar2,
   parent_itemkey in varchar2,
   parent_context in varchar2);
```

Java Syntax

```
public static boolean setItemParent
  (WFContext wCtx,
   String itemType,
   String itemKey,
   String parentItemType,
   String parentItemKey,
   String parentContext)
```

Description

Defines the parent/child relationship for a master process and a detail process. This API must be called by any detail process spawned from a master process to define the parent/child relationship between the two processes. You make a call to this API after you call the *CreateProcess* API, but before you call the *StartProcess* API for the detail process.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype or itemType
A valid item type.

itemkey or itemKey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the child process.

> **Note:** The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

parent_itemtype or parentItemType
A valid item type for the parent process.

parent_itemkey or parentItemKey
A string generated from the application object's primary key to uniquely identify the item within the parent item type. The parent item type and key together identify the parent process.

> **Note:** The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

parent_context or parentContext
If the parent process contains more than one Wait for Flow activity, set this parameter to the activity label name for the Wait for Flow activity node that corresponds to this detail process. If the parent process contains only one Wait for Flow activity, you can leave the parent context null.

ItemStatus

PL/SQL Syntax

```
procedure ItemStatus
  (itemtype in varchar2,
   itemkey in varchar2,
   status out varchar2,
   result out varchar2);
```

Java Syntax

```
public static WFTwoDArray itemStatus
  (WFContext wCtx,
   String itemType,
   String itemKey)
```

Description

Returns the status and result for the root process of the specified item instance. Possible values returned for the status are: `ACTIVE`, `COMPLETE`, `ERROR`, or `SUSPENDED`. If the root process does not exist, then the item key does not exist and will thus cause the procedure to raise an exception.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemtype
A valid item type.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the item instance.

Example

Example

The following code excerpt shows an example of how to call *itemStatus()* in a Java program. The example code is from the `WFTest.java` program.

```
// get status and result for this item
dataSource = WFEngineAPI.itemStatus(ctx, iType, iKey);
System.out.print("Status and result for " + iType + "/" +
    iKey + " = ");
displayDataSource(ctx, dataSource);
```

getProcessStatus

Java Syntax

```
public static WFTwoDArray getProcessStatus
  (WFContext wCtx,
   String itemType,
   String itemKey,
   BigDecimal process)
```

Description

Returns the process status for the given item type instance as a two-dimensional data object.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemType
A valid item type.

itemKey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

process
A process instance ID for the item type. If the instance ID is unknown, you can simply provide any negative number and the method will return the process status for the root process.

Workflow Function APIs

The WFFunctionAPI Java class is the abstract class from which the Java procedures for all external Java function activities are derived. This class contains methods for accessing item type and activity attributes, as well as the *execute()* method which forms the main entry point function of the external Java function activity being implemented.

The WFFunctionAPI class is stored in the `oracle.apps.fnd.wf` Java package. The following list shows the APIs available in this class.

> **Important:** Java is case-sensitive and all Java method names begin with a lower case letter to follow Java naming conventions.

- loadItemAttributes, page 2-61
- loadActivityAttributes, page 2-61
- getActivityAttr, page 2-62
- getItemAttr, page 2-63
- setItemAttrValue, page 2-63
- execute, page 2-63

Related Topics

Standard API for Java Procedures Called by Function Activities, *Oracle Workflow Developer's Guide*

Function Activity, *Oracle Workflow Developer's Guide*

loadItemAttributes

Java Syntax

```
public void loadItemAttributes
  (WFContext pWCtx) throws SQLException
```

Description

Retrieves the item attributes from the database for the item type from which the external Java function was called. The item attributes are not loaded by default due to the performance impact that could occur if the item type contains a large number of item attributes. You can use this method to load the item attributes explicitly before accessing them in your function.

If a database access error occurs, this method throws a SQLException.

Arguments (input)

pWCtx
Workflow context information. See: Oracle Workflow Context, page 2-4.

loadActivityAttributes

Java Syntax

```
public void loadActivityAttributes
  (WFContext pWCtx,
   String iType,
   String iKey,
   BigDecimal actid) throws SQLException
```

Description

Retrieves the activity attributes from the database for the specified activity. This method is called by default when the function activity is instantiated and before the *execute()* function is called.

If a database access error occurs, this method throws a SQLException.

Arguments (input)

pWCtx
Workflow context information. See: Oracle Workflow Context, page 2-4.

iType
A valid item type.

iKey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process. See: CreateProcess, page 2-16.

actid
An activity instance ID.

getActivityAttr

Java Syntax

```
public WFAttribute getActivityAttr
  (String aName)

public WFAttribute getActivityAttr
  (WFContext pWCtx,
   String aName) throws SQLException
```

Description

There are two implementations of *getActivityAttr()*. These methods return the activity attribute information for the specified activity attribute.

- If you call *getActivityAttr(String aName)* with only the activity attribute name, this method returns the activity attribute value but does not attempt to resolve any reference to an item attribute. If an activity attribute does point to an item attribute, this method returns the internal name of the item attribute. With the item attribute name, you can then perform additional processing based on the item attribute.

  For example, if you want to write information back to the item attribute, you can first use *getActivityAttr(String aName)* to retrieve the item attribute name. Then use *setItemAttrValue(WFContext pWCtx, WFAttribute pAttr)* to set the item attribute value, which also becomes the activity attribute value. See: setItemAttrValue, page 2-63.

- If you call *getActivityAttr(WFContext pWCtx, String aName)* with both the Workflow context and the activity attribute name, this method returns the activity attribute, and if the activity attribute points to an item attribute, the method attempts to resolve the reference by retrieving the value of that item attribute. You can use *getActivityAttr(WFContext pWCtx, String aName)* when you want to obtain the actual activity attribute value, and you do not need to know which item attribute it references. This method attempts to resolve the reference within the previously loaded item attributes, or if the item attributes have not been loaded, the method calls *loadItemAttributes(WFContext pWCtx)* to load them. See: loadItemAttributes, page 2-61.

  If a database access error occurs, this method throws a SQLException.

Arguments (input)

pWCtx
Workflow context information. Required for the second method only. See: Oracle Workflow Context, page 2-4.

aName
The internal name of an activity attribute.

getItemAttr

Java Syntax

```
public WFAttribute getItemAttr
  (String aName)
```

Description

Returns the item attribute information for the specified item attribute.

Arguments (input)

aName
The internal name of an item attribute.

setItemAttrValue

Java Syntax

```
public void setItemAttrValue
  (WFContext pWCtx,
   WFAttribute pAttr)
  throws NumberFormatException, WFException
```

Description

Sets the value of the specified item attribute in the database.

This method throws a NumberFormatException if it cannot convert the value to the appropriate format for an attribute of type number or date. The method throws a WFException if it encounters an error while setting an attribute of type document or text.

Arguments (input)

pWCtx
Workflow context information. See: Oracle Workflow Context, page 2-4.

pAttr
The attribute information for an item attribute.

execute

Java Syntax

```
public abstract boolean execute
  (WFContext pWCtx)
```

Description

This abstract method is implemented by the extending class and forms the main entry point function of the external Java function activity being implemented. See: Standard API for Java Procedures Called by Function Activities, *Oracle Workflow Developer's Guide*.

Arguments (input)

pWCtx
Workflow context information. See: Oracle Workflow Context, page 2-4.

Workflow Attribute APIs

The WFAttribute Java class contains descriptive information for an item or activity attribute, including the internal name of the attribute, attribute value, attribute data type, format information, and default value type. The attribute value is stored as an Object type. This class also contains methods for accessing the attribute information, which can be called by a Java application or the Java procedure for an external Java function activity.

The WFAttribute class is stored in the `oracle.apps.fnd.wf` Java package. The following list shows the APIs available in this class.

> **Important:** Java is case-sensitive and all Java method names, except the constructor method names, begin with a lower case letter to follow Java naming conventions.

- WFAttribute, page 2-65
- value, page 2-66
- getName, page 2-66
- getValue, page 2-66
- getType, page 2-67
- getFormat, page 2-67
- getValueType, page 2-67
- toString, page 2-67
- compareTo, page 2-68

WFAttribute Class Constants

The WFAttribute class contains several constants. The following table shows the constants that can be used to represent the data type of an attribute.

Data Type Constants

Constant Variable Declaration	Constant Value
public static final String TEXT	"TEXT"
public static final String NUMBER	"NUMBER"
public static final String DATE	"DATE"
public static final String LOOKUP	"LOOKUP"
public static final String FORM	"FORM"
public static final String URL	"URL"
public static final String DOCUMENT	"DOCUMENT"
public static final String ROLE	"ROLE"
public static final String EVENT	"EVENT"

The following table shows the constants that can be used to represent the type of the default value for an attribute. The default value can be either a constant or, for an activity attribute, a reference to an item type attribute.

Default Value Type Constants

Constant Variable Declaration	Constant Value
public static final String CONSTANT	"CONSTANT"
public static final String ITEMATTR	"ITEMATTR"

Related Topics

Standard API for Java Procedures Called by Function Activities, *Oracle Workflow Developer's Guide*

WFAttribute

Java Syntax

```
public WFAttribute()

public WFAttribute
  (String pName
   String pType,
   Object pValue,
   String pValueType)
```

Description

There are two constructor methods for the WFAttribute class. The first constructor creates a new WFAttribute object. The second constructor creates a new WFAttribute object and initializes it with the specified attribute name, attribute type, value, and value type.

Arguments (input)

pName
The internal name of an item or activity attribute. Required for the second method only.

pType
The data type of the attribute. Required for the second method only.

pValue
The attribute value. Required for the second method only.

pValueType
The type of the default value for the attribute. The default value can be either a constant or, for an activity attribute, a reference to an item type attribute. Required for the second method only.

value

Java Syntax

```
public void value
  (Object pValue)
```

Description

Sets the value of the item or activity attribute within a WFAttribute object. The value must be cast to the Object type.

> **Important:** Using *value()* to set the attribute value within a WFAttribute object does not set the attribute value in the database. To set the value of an item attribute in the database, use *WFFunctionAPI.setItemAttrValue()*. See: setItemAttrValue, page 2-63.

Arguments (input)

pValue
The attribute value.

getName

Java Syntax

```
public String getName()
```

Description

Returns the internal name of the item or activity attribute.

getValue

Java Syntax

```
public Object getValue()
```

Description

Returns the value of the item or activity attribute as type Object.

getType

Java Syntax

```
public String getType()
```

Description

Returns the data type of the item or activity attribute. See: Attribute Types, *Oracle Workflow Developer's Guide*.

getFormat

Java Syntax

```
public String getFormat()
```

Description

Returns the format string for the item or activity attribute, such as the length for an attribute of type text or the format mask for an attribute of type number or date. See: To Define an Item Type or Activity Attribute, *Oracle Workflow Developer's Guide*.

getValueType

Java Syntax

```
public String getValueType()
```

Description

Returns the type of the default value for the item or activity attribute. The default value can be either a constant or, for an activity attribute, a reference to an item type attribute. See: To Define an Item Type or Activity Attribute, *Oracle Workflow Developer's Guide*.

toString

Java Syntax

```
public String toString()
```

Description

Returns the internal name and the value of the item or activity attribute as a string in the following format:

<name>=<value>

This method overrides the *toString()* method in the Object class.

compareTo

Java Syntax

```
public int compareTo
  (String pValue) throws Exception
```

Description

Compares the value of the item or activity attribute with the specified value. *compareTo()* returns `0` if the two values are equal, `-1` if the attribute value is less than the specified value, or `1` if the attribute value is greater than the specified value.

This method throws an Exception if it cannot convert the specified value to the appropriate format for an attribute of type number or date.

Arguments (input)

pValue
The test value to compare to the attribute value.

Workflow Core APIs

PL/SQL procedures called by function activities can use a set of core Oracle Workflow APIs to raise and catch errors.

When a PL/SQL procedure called by a function activity either raises an unhandled exception, or returns a result beginning with `'ERROR:'`, the Workflow Engine sets the function activity's status to `ERROR` and sets the columns `ERROR_NAME`, `ERROR_MESSAGE`, and `ERROR_STACK` in the table WF_ITEM_ACTIVITY_STATUSES to reflect the error.

The columns `ERROR_NAME` and `ERROR_MESSAGE` get set to either the values returned by a call to *WF_CORE.RAISE()*, or to the SQL error name and message if no call to *RAISE()* is found. The column `ERROR_STACK` gets set to the contents set by a call to *WF_CORE.CONTEXT()*, regardless of the error source.

> **Note:** The columns `ERROR_NAME`, `ERROR_MESSAGE`, and `ERROR_STACK` are also defined as item type attributes for the System: Error predefined item type. You can reference the information in these columns from the error process that you associate with an activity. See: Error Handling for Workflow Processes, *Oracle Workflow Developer's Guide*.

The following APIs can be called by an application program or workflow function in the runtime phase to handle error processing. These APIs are stored in the PL/SQL package called WF_CORE.

- RAISE, page 2-71
- CONTEXT, page 2-74
- TRANSLATE, page 2-76

Related Topics

Standard API for PL/SQL Procedures Called by Function Activities, *Oracle Workflow Developer's Guide*

CLEAR

Syntax

```
procedure CLEAR;
```

Description

Clears the error buffers.

Related Topics

GET_ERROR, page 2-69

GET_ERROR

Syntax

```
procedure GET_ERROR
  (err_name out varchar2,
   err_message out varchar2
   err_stack out varchar2);
```

Description

Returns the name of a current error message and the token substituted error message. Also clears the error stack. Returns null if there is no current error.

Example

Example

```
/* Handle unexpected errors in your workflow code by raising
 * WF_CORE exceptions.  When calling any public Workflow API,
 * include an exception handler to deal with unexpected
 * errors.*/

declare
  errname varchar2(30);
  errmsg varchar2(2000);
  errstack varchar2(32000);

begin
  ...
  Wf_Engine.CompleteActivity(itemtype, itemkey, activity,
    result_code);
  ...

exception
  when others then
    wf_core.get_error(err_name, err_msg, err_stack);
    if (err_name is not null) then
      wf_core.clear;
      -- Wf error occurred.  Signal error as appropriate.
    else
      -- Not a wf error.  Handle otherwise.
    end if;
end;
```

Related Topics

CLEAR, page 2-69

TOKEN

Syntax

```
procedure TOKEN
  (token_name in varchar2,
   token_value in varchar2);
```

Description

Defines an error token and substitutes it with a value. Calls to *TOKEN()* and *RAISE()* raise predefined errors for Oracle Workflow that are stored in the WF_RESOURCES table. The error messages contain tokens that need to be replaced with relevant values when the error message is raised. This is an alternative to raising PL/SQL standard exceptions or custom-defined exceptions.

Arguments (input)

token_name
Name of the token.

token_value
Value to substitute for the token.

Related Topics

RAISE, page 2-71

CONTEXT, page 2-74

RAISE

Syntax

```
procedure RAISE
  (name in varchar2);
```

Description

Raises an exception to the caller by supplying a correct error number and token substituted message for the name of the error message provided.

Calls to *TOKEN()* and *RAISE()* raise predefined errors for Oracle Workflow that are stored in the WF_RESOURCES table. The error messages contain tokens that need to be replaced with relevant values when the error message is raised. This is an alternative to raising PL/SQL standard exceptions or custom-defined exceptions.

Error messages for Oracle Workflow are initially defined in message files (`.msg`). The message files are located in the *ORACLE_HOME*`/wf/res/<`*language*`>` directory for standalone Oracle Workflow or in the `$FND_TOP/import/<`*language*`>` directory for Oracle Applications. During the installation of Oracle Workflow, a program called Workflow Resource Generator takes the designated message files and imports the messages into the WF_RESOURCES table.

> **Note:** If you want to use custom error messages, you can define your messages in `.msg` files, load them to the WF_RESOURCES table, and then raise them using *RAISE()*. A custom error message must have an error number of 90000 or higher.

Arguments (input)

name
Internal name of the error message as stored in the table WF_RESOURCES.

To run the Workflow Resource Generator:

The standalone version of Oracle Workflow provides scripts to run the Workflow Resource Generator. In Oracle Applications, run the Workflow Resource Generator as a concurrent program.

For standalone Oracle Workflow:

To run the Workflow Resource Generator, run the `wfresgen.sh` script on UNIX or the `wfresgen.bat` script on Windows. These scripts are located in the *ORACLE_HOME*`/wf/admin` directory.

To upload seed data from a message source file (`.msg`) to the database table WF_RESOURCES, enter the following command at your operating system prompt.

- On Unix:

```
wfresgen.sh /mode loadresourcetodb  [/debug true] [/validate on]
/user <user> /connectstring <connectstring> /language <language>
/messagefile <message file>
```

- On Windows:

```
wfresgen.bat /mode loadresourcetodb  [/debug true] [/validate on]
 /user <user> /connectstring <connectstring> /language
<language> /messagefile <message file>
```

Specify the following parameters for the script:

- `/mode` - Specify the `loadresourcetodb` mode to upload seed data from a message source file to the database table WF_RESOURCES.
- `/debug` - Optionally include this parameter with the value `true` to report more extensive debugging information in the program output.
- `/validate` - Optionally include this parameter with the value `on` to validate the message source file against the database.
- `/user` - Specify the user name of your Oracle Workflow database account.
- `/connectstring` - Specify the Oracle Net connect string for the database.
- `/language` - Specify the language of the seed data to load, using a language abbreviation supported by the Oracle Database. For example, specify `JA` for Japanese. See: Locale Data, *Oracle Database Globalization Support Guide*.
- `/messagefile` - Specify the full path and name of the message source file you want to upload. You can include this parameter multiple times to specify multiple message source files. For example:

```
/messagefile <message file1> /messagefile <message file2>
```

After starting, the Workflow Resource Generator prompts you to enter the password for your Oracle Workflow database account.

Example

The following command shows an example of how to start the Workflow Resource Generator on Windows.

```
wfresgen.bat /mode loadresourcetodb /user OWF_MGR /connectstring
"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=wfhost.oracle.com)
(PORT=1521))(CONNECT_DATA=(SERVER=DEDICATED)
(SERVICE_NAME=orawf)))" /language US /messagefile
D:\Temp\wffile1.msg /messagefile D:\Temp\wffile2.msg
```

For Oracle Applications:

1. The Workflow Resource Generator program is registered as a concurrent program. You can run the Workflow Resource Generator concurrent program from the Submit Requests form or from the command line.
2. To run the concurrent program from the Submit Requests form, navigate to the Submit Requests form.

   Note: Your system administrator needs to add this concurrent program to a request security group for the responsibility that you want to run this program from. See: Overview of Concurrent

Programs and Requests, *Oracle Applications System Administrator's Guide*.

3. Submit the Workflow Resource Generator concurrent program as a request. See: Running Reports and Programs, *Oracle Applications User's Guide*.
4. In the Parameters window, enter values for the following parameters:

   Destination Type
   Specify "`Database`", to upload seed data to the database table WF_RESOURCES from a source file (`.msg`), or "`File`", to generate a resource file from a source file.

   Destination
   If you specify "`File`" for Destination Type, then enter the full path and name of the resource file you wish to generate. If you specify "`Database`" for Destination Type, then the program automatically uses the current database account as its destination.

   Source
   Specify the full path and name of your source file.
5. Choose OK to close the Parameters window.
6. When you finish modifying the print and run options for this request, choose Submit to submit the request.
7. Rather than use the Submit Requests form, you can also run the Workflow Resource Generator concurrent program from the command line using one of two commands. To generate a resource file from a source file, type:

   `WFRESGEN` *`apps/pwd`* `0 Y FILE` *`res_file source_file`*

   To upload seed data to the database table WF_RESOURCES from a source file, type:

   `WFRESGEN` *`apps/pwd`* `0 Y DATABASE` *`source_file`*

   Replace *`apps/pwd`* with the username and password to the `APPS` schema, replace *`res_file`* with the file specification of a resource file, and replace *`source_file`* with the file specification of a source file (`.msg`). A file specification is specified as:

   `@<application_short_name>:[<dir>/.../]file.ext`

   or

   `<native path>`

Related Topics

TOKEN, page 2-70

CONTEXT, page 2-74

CONTEXT

Syntax

```
procedure CONTEXT
  (pkg_name  IN VARCHAR2,
   proc_name IN VARCHAR2,
   arg1      IN VARCHAR2 DEFAULT '*none*',
   arg2      IN VARCHAR2 DEFAULT '*none*',
   arg3      IN VARCHAR2 DEFAULT '*none*',
   arg4      IN VARCHAR2 DEFAULT '*none*',
   arg5      IN VARCHAR2 DEFAULT '*none*');
```

Description

Adds an entry to the error stack to provide context information that helps locate the source of an error. Use this procedure with predefined errors raised by calls to *TOKEN()* and *RAISE(),* with custom-defined exceptions, or even without exceptions whenever an error condition is detected.

Arguments (input)

pkg_name
Name of the procedure package.

proc_name
Procedure or function name.

arg1
First IN argument.

argn
nth IN argument.

Example

Example

```
/*PL/SQL procedures called by function activities can use the
 * WF_CORE APIs to raise and catch errors the same way the
 * Workflow Engine does. */

package My_Package is

procedure MySubFunction(
  arg1 in varchar2,
  arg2 in varchar2)
is
...
begin
  if (<error condition>) then
    Wf_Core.Token('ARG1', arg1);
    Wf_Core.Token('ARG2', arg2);
    Wf_Core.Raise('ERROR_NAME');
  end if;
  ...

exception
  when others then
    Wf_Core.Context('My_Package', 'MySubFunction', arg1, arg2);
    raise;
end MySubFunction;

procedure MyFunction(
  itemtype in varchar2,
  itemkey in varchar2,
  actid in number,
  funcmode in varchar2,
  result out varchar2)
is
...
begin
  ...
  begin
    MySubFunction(arg1, arg2);
  exception
    when others then
      if (Wf_Core.Error_Name = 'ERROR_NAME') then
        -- This is an error I wish to ignore.
        Wf_Core.Clear;
      else
        raise;
      end if;
   end;
   ...
exception
  when others then
    Wf_Core.Context('My_Package', 'MyFunction', itemtype, itemkey,
                    to_char(actid), funcmode);
    raise;
end MyFunction;
```

Related Topics

TOKEN, page 2-70

RAISE, page 2-71

TRANSLATE

Syntax

```
function TRANSLATE
  (tkn_name IN VARCHAR2)
  return VARCHAR2;
```

Description

Translates the string value of a token by returning the value for the token as defined in WF_RESOURCES for your language setting.

Arguments (input)

tkn_name
Token name.

Workflow Purge APIs

The following APIs can be called by an application program or workflow function in the runtime phase to purge obsolete runtime and design data. These APIs are defined in the PL/SQL package called WF_PURGE.

WF_PURGE can be used to purge obsolete runtime data for completed items and processes, and to purge design information for obsolete activity versions that are no longer in use and expired users and roles. You may want to periodically purge this obsolete data from your system to increase performance.

A PL/SQL variable called "`persistence_type`"in the WF_PURGE package defines how most of the WF_PURGE APIs behave, with the exception of TotalPerm(). When the variable is set to `TEMP`, the WF_Purge APIs only purge data associated with Temporary item types, whose persistence, in days, has expired. The `persistence_type` variable is set to `TEMP` by default and should not be changed. If you need to purge runtime data for item types with Permanent persistence, you should use the procedure TotalPerm(). See: Persistence Type, *Oracle Workflow Developer's Guide*.

> **Important:** You cannot run any WF_PURGE API for a future end date. By entering a future end date, you may inadvertently violate the persistence period for Temporary item types. The WF_PURGE APIs will display an error message if you enter a future end date.

The three most commonly used procedures are:

- **WF_PURGE.ITEMS** - purge all runtime data associated with completed items, their processes, and notifications sent by them.
- **WF_PURGE.ACTIVITIES** - purge obsolete design versions of activities that are no longer in use by any item.

- **WF_PURGE.TOTAL** - purge both item data and activity design data.

The other auxiliary routines purge only certain tables or classes of data, and can be used in circumstances where a full purge is not desired.

The complete list of purge APIs is as follows:

- Items, page 2-77
- Activities, page 2-78
- Notifications, page 2-79
- Total, page 2-80
- TotalPERM, page 2-81
- Directory, page 2-82

> **Note:** The *AdHocDirectory()* API from earlier versions of Oracle Workflow is replaced by the *Directory()* API. The current version of Oracle Workflow still recognizes the *AdHocDirectory()* API for backward compatibility, but moving forward, you should only use the new *Directory()* API where appropriate.

In Oracle Applications, you can also use the "Purge Obsolete Workflow Runtime Data" concurrent program to purge obsolete item type runtime status information. See: Purge Obsolete Workflow Runtime Data, page 2-83.

In standalone Oracle Workflow, you can use the standalone Oracle Workflow Manager component available through Oracle Enterprise Manager to submit and manage Workflow purge database jobs. For more information, please refer to the Oracle Workflow Manager online help.

Related Topics

Standard API for PL/SQL Procedures Called by Function Activities, *Oracle Workflow Developer's Guide*

Purging for Performance, *Oracle Workflow Administrator's Guide*

Items

Syntax

```
procedure Items
  (itemtype in varchar2 default null,
   itemkey in varchar2 default null,
   enddate in date default sysdate,
   docommit in boolean default TRUE,
   force in boolean default FALSE);
```

Description

Deletes all items for the specified item type, and/or item key, and end date, including process status information, notifications, and any comments associated with these notifications. In Oracle Applications, any electronic signature information associated with these notifications is deleted as well. Deletes from the tables

WF_NOTIFICATIONS, WF_COMMENTS, WF_DIG_SIGS, WF_ITEM_ACTIVITY_STATUSES, WF_ITEM_ATTRIBUTE_VALUES and WF_ITEMS.

Arguments (input)

itemtype
Item type to delete. Leave this argument null to delete all item types.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. If null, the procedure purges all items in the specified item type.

enddate
Specified date to delete up to.

docommit
Specify `TRUE` or `FALSE` to indicate whether to commit data while purging. If you want *Items()* to commit data as it purges to reduce rollback size and improve performance, specify `TRUE`. If you do not want to perform automatic commits, specify `FALSE`. Defaults to `TRUE`.

force
Specify `TRUE` or `FALSE` to indicate whether to delete records for child items that have ended, even if the corresponding parent item does not yet have an end date. Defaults to `FALSE`.

Activities

Syntax

```
procedure Activities
  (itemtype in varchar2 default null,
   name in varchar2 default null,
   enddate in date default sysdate);
```

Description

Deletes old design versions of eligible activities from the tables WF_ACTIVITY_ATTR_VALUES, WF_ACTIVITY_TRANSITIONS, WF_PROCESS_ACTIVITIES, WF_ACTIVITY_ATTRIBUTES_TL, WF_ACTIVITY_ATTRIBUTES, WF_ACTIVITIES_TL, and WF_ACTIVITIES that are associated with the specified item type, have an END_DATE less than or equal to the specified end date, and are not referenced by an existing item as either a process or activity.

> **Note:** You should call *Items()* before calling *Activities()* to avoid having obsolete item references prevent obsolete activities from being deleted.

Arguments (input)

itemtype
Item type associated with the activities you want to delete. Leave this argument null to delete activities for all item types.

name
Internal name of activity to delete. Leave this argument null to delete all activities for the specified item type.

enddate
Specified date to delete up to.

Notifications

Syntax

```
procedure Notifications
  (itemtype in varchar2 default null,
   enddate in date default sysdate,
   docommit in boolean default TRUE);
```

Description

Deletes old eligible notifications from the tables WF_NOTIFICATION_ATTRIBUTES and WF_NOTIFICATIONS that are associated with the specified item type, have an END_DATE less than or equal to the specified end date, and are not referenced by an existing item. Also, any comments associated with these notifications are deleted from the WF_COMMENTS table. In Oracle Applications, any electronic signature information associated with these notifications is deleted from the WF_DIG_SIGS table as well. You can use this procedure to delete notifications that are not associated with any work item, such as notifications that were sent by calling *WF_NOTIFICATION.Send()* rather than through a workflow process.

> **Note:** You should call *Items()* before calling *Notifications()* to avoid having obsolete item references prevent obsolete notifications from being deleted.

Arguments (input)

itemtype
Item type associated with the notifications you want to delete. Leave this argument null to delete notifications for all item types.

enddate
Specified date to delete up to.

docommit
Specify `TRUE` or `FALSE` to indicate whether to commit data while purging. If you want *Notifications()* to commit data as it purges to reduce rollback size and improve performance, specify `TRUE`. If you do not want to perform automatic commits, specify `FALSE`. Defaults to `TRUE`.

Total

Syntax

```
procedure Total
  (itemtype in varchar2 default null,
   itemkey in varchar2 default null,
   enddate in date default sysdate,
   docommit in boolean default TRUE,
   runtimeonly in boolean default FALSE.
   transactiontype in varchar2 default null,
   transactionsubtype in varchar2 default null);
```

Description

Deletes all eligible obsolete runtime item type data that is associated with the specified item type and has an END_DATE less than or equal to the specified end date. In Oracle Applications, this procedure also deletes any Oracle XML Gateway transaction information associated with the items being purged.

If the RUNTIMEONLY parameter is set to TRUE, *Total()* deletes only runtime data associated with work items. However, if the RUNTIMEONLY parameter is set to FALSE, *Total()* also deletes these types of data:

- All eligible obsolete activity design data that is associated with the specified item type and has an END_DATE less than or equal to the specified end date. See: Activities, page 2-78.
- Expired users and roles in the Workflow local tables that are no longer in use. See: Directory, page 2-82.
- All eligible notifications that are associated with the specified item type, have an END_DATE less than or equal to the specified end date, and are not referenced by an existing item. See: Notifications, page 2-79.
- For Oracle Applications only, Oracle XML Gateway transaction information that is not associated with any existing work item. This information is purged using the ECX_PURGE.Purge_Items API. See: *Oracle XML Gateway User's Guide*.

Because *Total()* purges additional design data and runtime data not associated with work items when you set the RUNTIMEONLY parameter to FALSE, it is more costly in performance than *Items()*. If you want to purge a specific item key, use *Items()*, or set the RUNTIMEONLY parameter to TRUE when you run *Total()* to enhance performance. Run *Total()* with the RUNTIMEONLY parameter set to FALSE as part of your routine maintenance during periods of low activity. See: Items, page 2-77.

Arguments (input)

itemtype
Item type associated with the obsolete data you want to delete. Leave this argument null to delete obsolete data for all item types.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. If null, the procedure purges all items in the specified itemtype.

enddate
Specified date to delete up to.

docommit
Specify TRUE or FALSE to indicate whether to commit data while purging. If you want *Total()* to commit data as it purges to reduce rollback size and improve performance, specify TRUE. If you do not want to perform automatic commits, specify FALSE. Defaults to TRUE.

runtimeonly
Specify TRUE to purge only obsolete runtime data associated with work items, or FALSE to purge all obsolete runtime data as well obsolete design data. Defaults to FALSE.

transactiontype
The Oracle XML Gateway transaction type to purge. Leave this argument null to purge the runtime data for all transaction types.

transactionsubtype
The Oracle XML Gateway transaction subtype to purge. The transaction subtype is a code for a particular transaction within the application specified by the transaction type. Leave this argument null to purge the runtime data for all transactions of the specified transaction type.

TotalPERM

Syntax

```
procedure TotalPERM
  (itemtype in varchar2 default null,
   itemkey in varchar2 default null,
   enddate in date default sysdate,
   docommit in boolean default TRUE,
   runtimeonly in boolean default FALSE);
```

Description

Deletes all eligible obsolete runtime data that is of persistence type 'PERM' (Permanent) and that is associated with the specified item type and has an END_DATE less than or equal to the specified end date. In Oracle Applications, this procedure also deletes any Oracle XML Gateway transaction information associated with the items being purged.

If the RUNTIMEONLY parameter is set to TRUE, *TotalPERM()* deletes only runtime data associated with work items. However, if the RUNTIMEONLY parameter is set to FALSE, *TotalPERM()* also deletes these types of data:

- All eligible obsolete activity design data that is associated with the specified item type and has an END_DATE less than or equal to the specified end date. See: Activities, page 2-78.
- Expired users and roles in the Workflow local tables that are no longer in use. See: Directory, page 2-82.
- All eligible notifications that are associated with the specified item type, have an END_DATE less than or equal to the specified end date, and are not referenced by an existing item. See: Notifications, page 2-79.

- For Oracle Applications only, Oracle XML Gateway transaction information that is not associated with any existing work item. This information is purged using the ECX_PURGE.Purge_Items API. See: *Oracle XML Gateway User's Guide*.

TotalPERM() is similar to *Total()* except that *TotalPERM()* deletes only items with a persistence type of 'PERM'. See: Total, page 2-80.

Arguments (input)

itemtype
Item type associated with the obsolete runtime data you want to delete. Leave this argument null to delete obsolete runtime data for all item types.

itemkey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. If null, the procedure purges all items in the specified itemtype.

enddate
Specified date to delete up to.

docommit
Specify TRUE or FALSE to indicate whether to commit data while purging. If you want *TotalPERM()* to commit data as it purges to reduce rollback size and improve performance, specify TRUE. If you do not want to perform automatic commits, specify FALSE. Defaults to TRUE.

runtimeonly
Specify TRUE to purge only obsolete runtime data associated with work items, or FALSE to purge all obsolete runtime data as well obsolete design data. Defaults to FALSE.

Directory

Syntax

```
procedure Directory
   (end_date in date default sysdate);
```

Description

Purges all users and roles in the WF_LOCAL_ROLES and WF_LOCAL_USER_ROLES tables whose expiration date is less than or equal to the specified end date and that are not referenced in any notification.

Note that although users and roles whose expiration date has passed do not appear in the seeded WF_USERS, WF_ROLES, and WF_USER_ROLES views, they are not removed from the Workflow local tables until you purge them using *Directory()*. You should periodically purge expired users and roles in order to improve performance.

Arguments (input)

end_date
Date to purge to.

Purge Obsolete Workflow Runtime Data Concurrent Program

If you are using the version of Oracle Workflow embedded in Oracle Applications, you can submit the Purge Obsolete Workflow Runtime Data concurrent program to purge obsolete runtime work item information, including status information and any associated notifications and Oracle XML Gateway transactions. Use the Submit Requests form in Oracle Applications to submit this concurrent program.

By default, this program purges obsolete runtime information associated with work items as well as obsolete design information, such as activities that are no longer in use and expired users and roles, and obsolete runtime information not associated with work items, such as notifications or Oracle XML Gateway transactions that were not handled through a workflow process. You can optionally choose to purge only core runtime information associated with work items for performance gain during periods of high activity, and purge all obsolete information as part of your routine maintenance during periods of low activity.

> **Note:** You can also use the Oracle Workflow Manager component of Oracle Applications Manager to submit and manage the Purge Obsolete Workflow Runtime Data concurrent program. For more information, please refer to the Oracle Applications Manager online help.

To Purge Obsolete Workflow Runtime Data:

1. Navigate to the Submit Requests form in Oracle Applications to submit the Purge Obsolete Workflow Runtime Data concurrent program. When you install and set up Oracle Applications and Oracle Workflow, your system administrator needs to add this concurrent program to a request security group for the responsibility that you want to run this program from. The executable name for this concurrent program is "Oracle Workflow Purge Obsolete Data" and its short name is `FNDWFPR`. See: Overview of Concurrent Programs and Requests, *Oracle Applications System Administrator's Guide*.
2. Submit the Purge Obsolete Workflow Runtime Data concurrent program as a request. See: Running Reports and Programs, *Oracle Applications User's Guide*.
3. In the Parameters window, enter values for the following parameters:

   Item Type
   Item type associated with the obsolete runtime data you want to delete. Leave this argument null to delete obsolete runtime data for all item types.

   Item Key
   A string generated from the application object's primary key. The string uniquely identifies the item within an item type. If null, the program purges all items in the specified item type.

   Age
   Minimum age of data to purge, in days, if the persistence type is set to '`TEMP`'. The default is `0`.

   Persistence Type
   Persistence type to be purged, either '`TEMP`' for Temporary or '`PERM`' for Permanent. The default is '`TEMP`'.

Core Workflow Only
Enter 'Y' to purge only obsolete runtime data associated with work items, or 'N' to purge all obsolete runtime data as well obsolete design data. The default is 'N'.

Transaction Type
The Oracle XML Gateway transaction type to purge. Leave this argument null to purge the runtime data for all transaction types.

Transaction Subtype
The Oracle XML Gateway transaction subtype to purge. The transaction subtype is a code for a particular transaction within the application specified by the transaction type. Leave this argument null to purge the runtime data for all transactions of the specified transaction type.

4. Choose OK to close the Parameters window.
5. When you finish modifying the print and run options for this request, choose Submit to submit the request.

Workflow Monitor APIs

Call the following APIs to retrieve an access key or to generate a complete URL to access various pages of the Workflow Monitor in standalone Oracle Workflow, or to access various pages of the administrator version of the Status Monitor in Oracle Applications with guest access. The Workflow Monitor APIs are defined in the PL/SQL package called WF_MONITOR.

- GetAccessKey, page 2-85
- GetDiagramURL, page 2-85
- GetEnvelopeURL, page 2-87
- GetAdvancedEnvelopeURL, page 2-88

Note: The GetURL API from earlier versions of Oracle Workflow is replaced by the GetEnvelopeURL and GetDiagramURL APIs. The functionality of the previous GetURL API correlates directly with the new GetDiagramURL. API. The current version of Oracle Workflow still recognizes the GetURL API, but moving forward, you should only use the two new APIs where appropriate.

Note: Oracle Workflow also provides Java methods for accessing the Status Monitor in Oracle Applications, which are defined in the Java class called oracle.apps.fnd.wf.monitor.webui.Monitor. For more information about these methods, refer to the Javadoc for the oracle.apps.fnd.wf.monitor.webui.Monitor class.

Related Topics

Providing Access to the Status Monitor from Applications, *Oracle Workflow Administrator's Guide*

Guest Access in PL/SQL, *Oracle Workflow Administrator's Guide*

GetAccessKey

Syntax

```
function GetAccessKey
  (x_item_type varchar2,
   x_item_key varchar2,
   x_admin_mode varchar2)
  return varchar2;
```

Description

Retrieves the access key password that controls access to the Workflow Monitor. Each process instance has separate access keys for running the Workflow Monitor in 'ADMIN' mode or 'USER' mode.

Arguments (input)

x_item_type
A valid item type.

x_item_key
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process to report on.

x_admin_mode
A value of `YES` or `NO`. `YES` directs the function to retrieve the access key password that runs the monitor in 'ADMIN' mode. `NO` retrieves the access key password that runs the monitor in 'USER' mode.

GetDiagramURL

Syntax

```
function GetDiagramURL
  (x_agent in varchar2,
   x_item_type in varchar2,
   x_item_key in varchar2,
   x_admin_mode in varchar2 default 'NO')
  return varchar2;
```

Description

Can be called by an application to return a URL that allows access to a status diagram in the Workflow Monitor in standalone Oracle Workflow with an attached access key password, or to the Status Diagram page in the Status Monitor in Oracle Applications with guest access.

- In standalone Oracle Workflow, the URL displays the diagram for a specific instance of a workflow process in the Workflow Monitor operating in either 'ADMIN' or 'USER' mode.

  The URL returned by the function *WF_MONITOR.GetDiagramURL()* looks as follows:

```
<webagent>/wf_monitor.html?x_item_type=<item_type>&x_item_key=
<item_key>&x_admin_mode=<YES or NO>&x_access_key=<access_key>
```

<webagent> represents the base URL of the Web agent configured for Oracle Workflow in your Web server. See: Setting Global User Preferences, *Oracle Workflow Administrator's Guide*.

`wf_monitor.html` is the name of the PL/SQL package procedure that generates the Workflow Monitor diagram of the process instance.

The `wf_monitor.html` procedure requires four arguments. *<item_type>* and *<item_key>* represent the internal name of the item type and the item key that uniquely identify an instance of a process. If *<YES or NO>* is `YES`, the monitor runs in 'ADMIN' mode and if `NO`, the monitor runs in 'USER' mode. *<access_key>* represents the access key password that determines whether the monitor is run in 'ADMIN' or 'USER' mode.

- In Oracle Applications, the URL displays the Status Diagram page for a specific instance of a workflow process in the administrator version of the Status Monitor, operating either with or without administrator privileges.

Arguments (input)

x_agent
The base Web agent string defined for Oracle Workflow or Oracle Self-Service Web Applications in your Web server. The base Web agent string should be stored in the WF_RESOURCES table, and looks something like:

`http://<server.com:portID>/<PLSQL_agent_path>`
When calling this function, your application must first retrieve the Web agent string from the WF_RESOURCES token `WF_WEB_AGENT` by calling *WF_CORE.TRANSLATE()*. See: Setting Global User Preferences, *Oracle Workflow Administrator's Guide* or Applications Web Agent, *Oracle Applications System Administrator's Guide*.

x_item_type
A valid item type.

x_item_key
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process to report on.

x_admin_mode
A value of `YES` or `NO`. `YES` directs the function to retrieve the access key password that runs the monitor in 'ADMIN' mode in standalone Oracle Workflow, or to grant administrator privileges to the user accessing the Status Monitor in Oracle Applications. `NO` directs the function to retrieve the access key password that runs the monitor in 'USER' mode in standalone Oracle Workflow, or to withhold administrator privileges from the user accessing the Status Monitor in Oracle Applications.

Example

Example
Following is an example of how you can call *GetDiagramUrl()*. This example returns a URL that displays the diagram page for a process instance identified by the item type `WFDEMO` and item key `10022`, in 'USER' mode or without administrator privileges:

```
URL := WF_MONITOR.GetDiagramURL
  (WF_CORE.Translate('WF_WEB_AGENT'),
   'WFDEMO',
   '10022',
   'NO');
```

Related Topics

TRANSLATE, page 2-76

GetEnvelopeURL

Syntax

```
function GetEnvelopeURL
  (x_agent in varchar2,
   x_item_type in varchar2,
   x_item_key in varchar2,
   x_admin_mode in varchar2 default 'NO')
  return varchar2;
```

Description

Can be called by an application to return a URL that allows access to the Workflow Monitor Notifications List in standalone Oracle Workflow with an attached access key password, or to the Participant Responses page in the Status Monitor in Oracle Applications with guest access.

- In standalone Oracle Workflow, the URL displays the Notifications List for a specific instance of a workflow process in the Workflow Monitor.

  The URL returned by the function *WF_MONITOR.GetEnvelopeURL()* looks as follows:

  <webagent>/wf_monitor.envelope?x_item_type=*<item_type>*&x_item_key=*<item_key>*&x_admin_mode=*<YES or NO>*&x_access_key=*<access_key>*

  <webagent> represents the base URL of the Web agent configured for Oracle Workflow in your Web server. See: Setting Global User Preferences, *Oracle Workflow Administrator's Guide.*

  wf_monitor.envelope is the name of the PL/SQL package procedure that generates the Workflow Monitor Notifications List for the process instance.

- In Oracle Applications, the URL displays the Participant Responses page for a specific instance of a workflow process in the administrator version of the Status Monitor, operating either with or without administrator privileges.

Arguments (input)

x_agent
The base Web agent string defined for Oracle Workflow or Oracle Self-Service Web Applications in your Web server. The base Web agent string should be stored in the WF_RESOURCES table, and looks something like:

http://*<server.com:portID>*/*<PLSQL_agent_path>*

When calling this function, your application must first retrieve the Web agent string from the WF_RESOURCES token `WF_WEB_AGENT` by calling *WF_CORE.TRANSLATE()*. See: Setting Global User Preferences, *Oracle Workflow Administrator's Guide* or Applications Web Agent, *Oracle Applications System Administrator's Guide*.

x_item_type
A valid item type.

x_item_key
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process to report on.

x_admin_mode
A value of `YES` or `NO`. `YES` directs the function to retrieve the access key password that runs the monitor in 'ADMIN' mode in standalone Oracle Workflow, or to grant administrator privileges to the user accessing the Status Monitor in Oracle Applications. `NO` directs the function to retrieve the access key password that runs the monitor in 'USER' mode in standalone Oracle Workflow, or to withhold administrator privileges from the user accessing the Status Monitor in Oracle Applications.

Related Topics

TRANSLATE, page 2-76

GetAdvancedEnvelopeURL

Syntax

```
function GetAdvancedEnvelopeURL
  (x_agent in varchar2,
   x_item_type in varchar2,
   x_item_key in varchar2,
   x_admin_mode in varchar2 default 'NO',
   x_options in varchar2 default null)
  return varchar2;
```

Description

Can be called by an application to return a URL that displays the Workflow Monitor Activities List in standalone Oracle Workflow with an attached access key password, or to the Activity History page in the Status Monitor in Oracle Applications with guest access.

- In standalone Oracle Workflow, the URL displays the Activities List for a specific instance of a workflow process in the Workflow Monitor. The Activities List allows you to apply advanced filtering options in displaying the list of activities for a process instance.

  The URL returned by the function *WF_MONITOR.GetAdvancedEnvelopeURL()* looks as follows if the `x_options` argument is null:

  <webagent>/wf_monitor.envelope?x_item_type=*<item_type>*&x_item_key=*<item_key>*&x_admin_mode=*<YES or NO>*&x_access_key=*<access_key>*&x_advanced=TRUE

<webagent> represents the base URL of the Web agent configured for Oracle Workflow in your Web server. See: Setting Global User Preferences, *Oracle Workflow Administrator's Guide.*

`wf_monitor.envelope` is the name of the PL/SQL package procedure that generates the Workflow Monitor Notifications List for the process instance.

- In Oracle Applications, the URL displays the Activity History page for a specific instance of a workflow process in the administrator version of the Status Monitor, operating either with or without administrator privileges. All activity type and activity status filtering options are automatically selected by default.

Arguments (input)

x_agent
The base Web agent string defined for Oracle Workflow or Oracle Self-Service Web Applications in your Web server. The base Web agent string should be stored in the WF_RESOURCES table, and looks something like:

`http://<server.com:portID>/<PLSQL_agent_path>`

When calling this function, your application must first retrieve the Web agent string from the WF_RESOURCES token `WF_WEB_AGENT` by calling *WF_CORE.TRANSLATE()*. See: Setting Global User Preferences, *Oracle Workflow Administrator's Guide* or Applications Web Agent, *Oracle Applications System Administrator's Guide*.

x_item_type
A valid item type.

x_item_key
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process to report on.

x_admin_mode
A value of `YES` or `NO`. `YES` directs the function to retrieve the access key password that runs the monitor in 'ADMIN' mode in standalone Oracle Workflow, or to grant administrator privileges to the user accessing the Status Monitor in Oracle Applications. `NO` directs the function to retrieve the access key password that runs the monitor in 'USER' mode in standalone Oracle Workflow, or to withhold administrator privileges from the user accessing the Status Monitor in Oracle Applications.

x_options
In standalone Oracle Workflow only, specify `'All'` if you wish to return a URL that displays the Activities List with all filtering options checked. If you leave this argument null, then a URL is returned that displays the Activities List with no filtering options checked. This allows you to append any specific options if you wish. The default is null.

> **Note:** The `x_options` parameter does not apply for the Status Monitor in Oracle Applications. When you access the Status Monitor with a URL from *GetAdvancedEnvelopeURL()*, all filtering options are always selected by default.

Related Topics

TRANSLATE, page 2-76

Workflow Status Monitor APIs

Call the following APIs to retrieve parameters for use with the self-service functions that provide access to the Status Monitor from Oracle Applications forms. You can use these APIs to help integrate other applications with the Status Monitor.

The Workflow Status Monitor PL/SQL APIs are defined in the PL/SQL package called WF_FWKMON.

- GetEncryptedAccessKey, page 2-90
- GetEncryptedAdminMode, page 2-91
- IsMonitorAdministrator, page 2-91

Related Topics

Providing Access to the Status Monitor from Applications, *Oracle Workflow Administrator's Guide*

Guest Access from Oracle E-Business Suite Forms, *Oracle Workflow Administrator's Guide*

GetEncryptedAccessKey

PL/SQL Syntax

```
function GetEncryptedAccessKey
  (itemType in varchar2,
   itemKey in varchar2,
   adminMode in varchar2 default 'N')
  return varchar2;
```

Description

Returns an encrypted access key password that controls access to the specified workflow process instance in the Status Monitor with the specified administrator mode. The administrator mode lets you determine whether the user who accesses the Status Monitor with this access key should have privileges to perform administrative operations in the Status Monitor.

Arguments (input)

itemType
A valid workflow item type.

itemKey
A string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the workflow process.

adminMode
Specify `'Y'` to grant administrator privileges to the user accessing the Status Monitor, or `'N'` to withhold administrator privileges from the user. The default is `'N'`.

GetEncryptedAdminMode

PL/SQL Syntax

```
function GetEncryptedAdminMode
  (adminMode in varchar2)
  return varchar2;
```

Description

Returns an encrypted value for the specified administrator mode. The administrator mode lets you determine whether a user accessing the Status Monitor should have privileges to perform administrative operations in the Status Monitor.

Arguments (input)

adminMode
Specify `'Y'` to grant administrator privileges to the user accessing the Status Monitor, or `'N'` to withhold administrator privileges from the user. The default is `'N'`.

IsMonitorAdministrator

PL/SQL Syntax

```
function IsMonitorAdministrator
  (userName in varchar2)
  return varchar2;
```

Description

Returns `'Y'` if the specified user has workflow administrator privileges, or `'N'` if the specified user does not have workflow administrator privileges. Workflow administrator privileges are assigned in the Workflow Configuration page. See: Setting Global User Preferences, *Oracle Workflow Administrator's Guide*.

For example, you can use this function to help determine what administrator mode to choose when calling *GetEncryptedAccesKey()* or *GetEncryptedAdminMode()* to retrieve parameters for use with the Status Monitor form functions.

Arguments (input)

userName
A valid user name.

Oracle Workflow Views

Public views are available for accessing workflow data. If you are using the version of Oracle Workflow embedded in Oracle Applications, these views are installed in the `APPS` account. If you are using the standalone version of Oracle Workflow, these views are installed in the same account as the Oracle Workflow server.

- WF_ITEM_ACTIVITY_STATUSES_V, page 2-92
- WF_NOTIFICATION_ATTR_RESP_V, page 2-93

- WF_RUNNABLE_PROCESSES_V, page 2-94
- WF_ITEMS_V, page 2-94

Note: These database views are public, meaning they are available for you to use for your custom data requirements. This description does not mean that any privileges for these views have been granted to `PUBLIC`.

WF_ITEM_ACTIVITY_STATUSES_V

This view contains denormalized information about a workflow process and its activities' statuses. Use this view to create custom queries and reports on the status of a particular item or process.

The following table describes the columns of the view.

WF_ITEM_ACTIVITY_STATUSES_V Columns

Name	Null?	Type
ROWID		ROWID
SOURCE		CHAR(1)
ITEM_TYPE		VARCHAR2(8)
ITEM_TYPE_DISPLAY_NAME		VARCHAR2(80)
ITEM_TYPE_DESCRIPTION		VARCHAR2(240)
ITEM_KEY		VARCHAR2(240)
USER_KEY		VARCHAR2(240)
ITEM_BEGIN_DATE		DATE
ITEM_END_DATE		DATE
ACTIVITY_ID		NUMBER
ACTIVITY_LABEL		VARCHAR2(30)
ACTIVITY_NAME		VARCHAR2(30)
ACTIVITY_DISPLAY_NAME		VARCHAR2(80)
ACTIVITY_DESCRIPTION		VARCHAR2(240)
ACTIVITY_TYPE_CODE		VARCHAR2(8)
ACTIVITY_TYPE_DISPLAY_NAME		VARCHAR2(80)
EXECUTION_TIME		NUMBER
ACTIVITY_BEGIN_DATE		DATE
ACTIVITY_END_DATE		DATE
ACTIVITY_STATUS_CODE		VARCHAR2(8)

Name	Null?	Type
ACTIVITY_STATUS_DISPLAY_NAME		VARCHAR2(80)
ACTIVITY_RESULT_CODE		VARCHAR2(30)
ACTIVITY_RESULT_DISPLAY_NAME		VARCHAR2(4000)
ASSIGNED_USER		VARCHAR2(30)
ASSIGNED_USER_DISPLAY_NAME		VARCHAR2(4000)
NOTIFICATION_ID		NUMBER
OUTBOUND_QUEUE_ID		RAW(16)
ERROR_NAME		VARCHAR2(30)
ERROR_MESSAGE		VARCHAR2(2000)
ERROR_STACK		VARCHAR2(4000)

Note: The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

WF_NOTIFICATION_ATTR_RESP_V

This view contains information about the Respond message attributes for a notification group. If you plan to create a custom voting activity, use this view to create the function that tallies the responses from the users in the notification group. See: Voting Activity, *Oracle Workflow Developer's Guide*.

The following table describes the columns of the view.

WF_NOTIFICATION_ATTR_RESP_V Columns

Name	Null?	Type
GROUP_ID	NOT NULL	NUMBER
RECIPIENT_ROLE	NOT NULL	VARCHAR2(30)
RECIPIENT_ROLE_DISPLAY_NAME		VARCHAR2(4000)
ATTRIBUTE_NAME	NOT NULL	VARCHAR2(30)
ATTRIBUTE_DISPLAY_NAME	NOT NULL	VARCHAR2(80)
ATTRIBUTE_VALUE		VARCHAR2(2000)
ATTRIBUTE_DISPLAY_VALUE		VARCHAR2(4000)
MESSAGE_TYPE	NOT NULL	VARCHAR2(8)
MESSAGE_NAME	NOT NULL	VARCHAR2(30)

WF_RUNNABLE_PROCESSES_V

This view contains a list of all runnable workflow processes in the ACTIVITIES table.

The following table describes the columns of the view.

WF_RUNNABLE_PROCESSES_V Columns

Name	Null?	Type
ITEM_TYPE	NOT NULL	VARCHAR2(8)
PROCESS_NAME	NOT NULL	VARCHAR2(30)
DISPLAY_NAME	NOT NULL	VARCHAR2(80)

WF_ITEMS_V

This view is a select-only version of the WF_ITEMS table.

The following table describes the columns of the view.

WF_ITEMS_V Columns

Name	Null?	Type
ITEM_TYPE	NOT NULL	VARCHAR2(8)
ITEM_KEY	NOT NULL	VARCHAR2(240)
USER_KEY		VARCHAR2(240)
ROOT_ACTIVITY	NOT NULL	VARCHAR2(30)
ROOT_ACTIVITY_VERSION	NOT NULL	NUMBER
OWNER_ROLE		VARCHAR2(30)
PARENT_ITEM_TYPE		VARCHAR2(8)
PARENT_ITEM_KEY		VARCHAR2(240)
PARENT_CONTEXT		VARCHAR2(2000)
BEGIN_DATE	NOT NULL	DATE
END_DATE		DATE

Note: The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

3

Directory Service APIs

This chapter describes the APIs for the Oracle Workflow directory service. The APIs consist of PL/SQL functions and procedures that you can use to access the directory service.

This chapter covers the following topics:

- Workflow Directory Service APIs
- Workflow LDAP APIs
- Workflow Local Synchronization APIs
- Workflow Role Hierarchy APIs
- Workflow Preferences API

Workflow Directory Service APIs

The following APIs can be called by an application program or a workflow function in the runtime phase to retrieve information about existing users and roles, as well as create and manage new ad hoc users and roles in the directory service. These APIs are defined in a PL/SQL package called WF_DIRECTORY.

- RemoveUsersFromAdHocRole, page 3-13
- SetAdHocUserStatus, page 3-13
- SetAdHocRoleStatus, page 3-14
- SetAdHocUserExpiration, page 3-14
- SetAdHocRoleExpiration, page 3-15
- SetAdHocUserAttr, page 3-15
- SetAdHocRoleAttr, page 3-17
- ChangeLocalUserName, page 3-18
- IsMLSEnabled, page 3-18

Important: If you implement Oracle Internet Directory integration, you must maintain your users only through Oracle Internet Directory. You must not create ad hoc users in the WF_LOCAL_ROLES table, because you risk discrepancies in your user information and unpredictable results if you use any tool other than Oracle Internet Directory to maintain users after integrating with Oracle Internet Directory. Consequently, if you implement Oracle Internet Directory integration, you must not use the *CreateAdHocUser()*, *SetAdHocUserStatus()*, *SetAdHocUserExpiration()*, or *SetAdHocUserAttr()* APIs in the WF_DIRECTORY package.

You can still use ad hoc roles, however, since Workflow roles are not maintained through Oracle Internet Directory.

Some directory service APIs use PL/SQL table composite datatypes defined in the WF_DIRECTORY package. The following table shows the column datatype definition for each PL/SQL table type.

PL/SQL Table Types in WF_DIRECTORY

PL/SQL Table Type	Column Datatype Definition
UserTable	varchar2(320)
RoleTable	varchar2(320)

Related Topics

Standard API for PL/SQL Procedures Called by Function Activities, *Oracle Workflow Developer's Guide*

GetRoleUsers

Syntax

```
procedure GetRoleUsers
  (role in varchar2,
   users out wF_DIRECTORY.UserTable);
```

Description

Returns a table of users for a given role.

> **Note:** A role can contain only individual users as its members. It cannot contain another role.

Arguments (input)

role
A valid role name.

GetUserRoles

Syntax

```
procedure GetUserRoles
  (user in varchar2,
   roles out WF_DIRECTORY.RoleTable);
```

Description

Returns a table of roles that a given user is assigned to.

Arguments (input)

user
A valid username.

GetRoleInfo

Syntax

```
procedure GetRoleInfo
  (Role in varchar2,
   Display_Name out varchar2,
   Email_Address out varchar2,
   Notification_Preference out varchar2,
   Language out varchar2,
   Territory out varchar2);
```

Description

Returns the following information about a role:

- Display name
- E-mail address
- Notification Preference (`'QUERY'`,`'MAILTEXT'`,`'MAILHTML'`,`'MAILATTH'`,`'MAILHTM2'`,`'SUMMARY'`, or, for Oracle Applications only, `'SUMHTML'`)
- Language
- Territory

Note: In Oracle Applications, for roles that are Oracle Applications users marked with an originating system of FND_USR or PER, the *GetRoleInfo()* procedure by default retrieves the language and territory values from the ICX: Language and ICX: Territory profile options for that Oracle Applications user.

However, if the WF_PREFERENCE resource token is defined and set to FND, then the *GetRoleInfo()* procedure obtains the language and territory values from the Oracle Workflow preferences table instead.

The WF_PREFERENCE resource token is not used in the standalone version of Oracle Workflow.

Arguments (input)

role
A valid role name.

GetRoleInfo2

Syntax

```
procedure GetRoleInfo2
  (Role in varchar2,
   Role_Info_Tbl out wf_directory.wf_local_roles_tbl_type);
```

Description

Returns the following information about a role in a PL/SQL table:

- Name
- Display name
- Description
- Notification preference ('QUERY','MAILTEXT','MAILHTML','MAILATTH','MAILHTM2','SUMMARY', or, for Oracle Applications only, 'SUMHTML')
- Language
- Territory
- E-mail address
- Fax
- Status
- Expiration date
- Originating system
- Originating system ID
- Parent originating system
- Parent originating system ID
- Owner tag
- Standard Who columns

Note: In Oracle Applications, for roles that are Oracle Applications users marked with an originating system of FND_USR or PER, the *GetRoleInfo2()* procedure by default retrieves the language and territory values from the ICX: Language and ICX: Territory profile options for that Oracle Applications user.

However, if the WF_PREFERENCE resource token is defined and set to FND, then the *GetRoleInfo2()* procedure obtains the language and territory values from the Oracle Workflow preferences table instead.

The WF_PREFERENCE resource token is not used in the standalone version of Oracle Workflow.

Arguments (input)

role
A valid role name.

IsPerformer

Syntax

```
function IsPerformer
  (user in varchar2,
   role in varchar2)
  return boolean;
```

Description

Returns TRUE or FALSE to identify whether a user is a performer, also known as a member, of a role.

Arguments (input)

user
A valid username.

role
A valid role name.

UserActive

Syntax

```
function UserActive
  (username in varchar2)
  return boolean;
```

Description

Determines if a user currently has a status of 'ACTIVE' and is available to participate in a workflow. Returns TRUE if the user has a status of 'ACTIVE', otherwise it returns FALSE.

Arguments (input)

username
A valid username.

GetUserName

Syntax

```
procedure GetUserName
  (p_orig_system in varchar2,
   p_orig_system_id in varchar2,
   p_name out varchar2,
   p_display_name out varchar2);
```

Description

Returns a Workflow display name and username for a user given the system information from the original user and roles repository.

Arguments (input)

p_orig_system
Code that identifies the original repository table.

p_orig_system_id
ID of a row in the original repository table.

GetRoleName

Syntax

```
procedure GetRoleName
  (p_orig_system in varchar2,
   p_orig_system_id in varchar2,
   p_name out varchar2,
   p_display_name out varchar2);
```

Description

Returns a Workflow display name and role name for a role given the system information from the original user and roles repository.

Arguments (input)

p_orig_system
Code that identifies the original repository table.

p_orig_system_id
ID of a row in the original repository table.

GetRoleDisplayName

Syntax

```
function GetRoleDisplayName
  (p_role_name in varchar2)
  return varchar2;
  pragma restrict_references(GetRoleDisplayName, WNDS, WNPS);
```

Description

Returns a Workflow role's display name given the role's internal name.

Arguments (input)

p_role_name
The internal name of the role.

CreateAdHocUser

Syntax

```
procedure CreateAdHocUser
  (name in out varchar2,
   display_name in out varchar2,
   language in varchar2 default null,
   territory in varchar2 default null,
   description in varchar2 default null,
   notification_preference in varchar2 default 'MAILHTML',
   email_address in varchar2 default null,
   fax in varchar2 default null,
   status in varchar2 default 'ACTIVE',
   expiration_date in date default null,
   parent_orig_system in varchar2 default null,
   parent_orig_system_id in number default null);
```

Description

Creates a user at runtime by creating a value in the WF_LOCAL_ROLES table with the user flag set to `Y`. This is referred to as an ad hoc user.

> **Important:** If you implement Oracle Internet Directory integration for standalone Oracle Workflow, you must maintain your users only through Oracle Internet Directory. You must not use the *CreateAdHocUser()* API to create new users in the WF_LOCAL_ROLES table, because you risk discrepancies in your user information and unpredictable results if you use any tool other than Oracle Internet Directory to maintain users after integrating with Oracle Internet Directory.

Arguments (input)

name
An internal name for the user. The internal name must be no longer than 320 characters. It is recommended that the internal name be all uppercase. This procedure checks that the name provided does not already exist in WF_USERS and returns an error

if the name already exists. If you do not provide an internal name, the system generates an internal name for you where the name contains a prefix of '~WF_ADHOC-' followed by a sequence number.

display_name
The display name of the user. This procedure checks that the display name provided does not already exist in WF_USERS and returns an error if the display name already exists. If you do not provide a display name, the system generates one for you where the display name contains a prefix of '~WF_ADHOC-' followed by a sequence number.

language
The value of the database NLS_LANGUAGE initialization parameter that specifies the default language-dependent behavior of the user's notification session. If null, the procedure resolves this to the language setting of your current session.

territory
The value of the database NLS_TERRITORY initialization parameter that specifies the default territory-dependent date and numeric formatting used in the user's notification session. If null, the procedure resolves this to the territory setting of your current session.

description
An optional description for the user.

notification_preference
Indicate how this user prefers to receive notifications: 'MAILTEXT','MAILHTML', 'MAILATTH','MAILHTM2','QUERY','SUMMARY', or, for Oracle Applications only, 'SUMHTML'. If null, the procedure sets the notification preference to 'MAILHTML'.

email_address
A optional electronic mail address for this user.

fax
An optional fax number for the user.

status
The availability of the user to participate in a workflow process. The possible statuses are 'ACTIVE','EXTLEAVE','INACTIVE', and 'TMPLEAVE'. If null, the procedure sets the status to 'ACTIVE'.

expiration_date
The date at which the user is no longer valid in the directory service.

parent_orig_system
An optional code for the originating system of an entity that you want to mark as being related to this user.

parent_orig_system_id
The primary key that identifies the parent entity in the parent originating system.

Related Topics

Setting Up an Oracle Workflow Directory Service, *Oracle Workflow Administrator's Guide*

CreateAdHocRole

Syntax

```
procedure CreateAdHocRole
  (role_name in out varchar2,
   role_display_name in out varchar2,
   language in varchar2 default null,
   territory in varchar2 default null,
   role_description in varchar2 default null,
   notification_preference in varchar2 default 'MAILHTML',
   role_users in varchar2 default null,
   email_address in varchar2 default null,
   fax in varchar2 default null,
   status in varchar2 default 'ACTIVE',
   expiration_date in date default null,
   parent_orig_system in varchar2 default null,
   parent_orig_system_id in number default null,
   owner_tag in varchar2 default null);
```

Description

Creates a role at runtime by creating a value in the WF_LOCAL_ROLES table with the user flag set to N. This is referred to as an ad hoc role.

> **Note:** A role can contain only individual users as its members. It cannot contain another role.

Arguments (input)

role_name
An internal name for the role. The internal name must be no longer than 320 characters. It is recommended that the internal name be all uppercase. This procedure checks that the name provided does not already exist in WF_ROLES and returns an error if the name already exists. If you do not provide an internal name, the system generates an internal name for you where the name contains a prefix of '~WF_ADHOC-' followed by a sequence number.

role_display_name
The display name of the role. This procedure checks that the display name provided does not already exist in WF_ROLES and returns an error if the display name already exists. If you do not provide a display name, the system generates one for you where the display name contains a prefix of '~WF_ADHOC-' followed by a sequence number.

language
The value of the database NLS_LANGUAGE initialization parameter that specifies the default language-dependent behavior of the user's notification session. If null, the procedure resolves this to the language setting of your current session.

territory
The value of the database NLS_TERRITORY initialization parameter that specifies the default territory-dependent date and numeric formatting used in the user's notification session. If null, the procedure resolves this to the territory setting of your current session.

role_description
An optional description for the role.

notification_preference
Indicate how this role receives notifications: 'MAILTEXT','MAILHTML','MAILATTH', 'MAILHTM2','QUERY','SUMMARY', or, for Oracle Applications only, 'SUMHTML'. If null, the procedure sets the notification preference to 'MAILHTML'.

role_users
Indicate the names of the users that belong to this role, using commas or spaces to delimit the list.

email_address
A optional electronic mail address for this role or a mail distribution list defined by your electronic mail system.

fax
An optional fax number for the role.

status
The availability of the role to participate in a workflow process. The possible statuses are ACTIVE, EXTLEAVE, INACTIVE, and TMPLEAVE. If null, the procedure sets the status to 'ACTIVE'.

expiration_date
The date at which the role is no longer valid in the directory service.

parent_orig_system
An optional code for the originating system of an entity that you want to mark as being related to this role.

parent_orig_system_id
The primary key that identifies the parent entity in the parent originating system.

owner_tag
A code to identify the program or application that owns the information for this role.

Related Topics

Setting Up an Oracle Workflow Directory Service, *Oracle Workflow Administrator's Guide*

CreateAdHocRole2

Syntax

```
procedure CreateAdHocRole2
  (role_name in out varchar2,
   role_display_name in out varchar2,
   language in varchar2 default null,
   territory in varchar2 default null,
   role_description in varchar2 default null,
   notification_preference in varchar2 default 'MAILHTML',
   role_users in WF_DIRECTORY.UserTable,
   email_address in varchar2 default null,
   fax in varchar2 default null,
   status in varchar2 default 'ACTIVE',
   expiration_date in date default null,
   parent_orig_system in varchar2 default null,
   parent_orig_system_id in number default null,
   owner_tag in varchar2 default null);
```

Description

Creates a role at runtime by creating a value in the WF_LOCAL_ROLES table with the user flag set to N. This is referred to as an ad hoc role. *CreateAdHocRole2()* accepts the list of users who belong to the role in the WF_DIRECTORY.UserTable format, which lets you include user names that contain spaces or commas.

> **Note:** A role can contain only individual users as its members. It cannot contain another role.

Arguments (input)

role_name
An internal name for the role. The internal name must be no longer than 320 characters. It is recommended that the internal name be all uppercase. This procedure checks that the name provided does not already exist in WF_ROLES and returns an error if the name already exists. If you do not provide an internal name, the system generates an internal name for you where the name contains a prefix of '~WF_ADHOC-' followed by a sequence number.

role_display_name
The display name of the role. This procedure checks that the display name provided does not already exist in WF_ROLES and returns an error if the display name already exists. If you do not provide a display name, the system generates one for you where the display name contains a prefix of '~WF_ADHOC-' followed by a sequence number.

language
The value of the database NLS_LANGUAGE initialization parameter that specifies the default language-dependent behavior of the user's notification session. If null, the procedure resolves this to the language setting of your current session.

territory
The value of the database NLS_TERRITORY initialization parameter that specifies the default territory-dependent date and numeric formatting used in the user's notification session. If null, the procedure resolves this to the territory setting of your current session.

role_description
An optional description for the role.

notification_preference
Indicate how this role receives notifications: 'MAILTEXT','MAILHTML','MAILATTH', 'MAILHTM2','QUERY','SUMMARY', or, for Oracle Applications only, 'SUMHTML'. If null, the procedure sets the notification preference to 'MAILHTML'.

role_users
The names of the users that belong to this role, as a table in the WF_DIRECTORY.UserTable format.

email_address
A optional electronic mail address for this role or a mail distribution list defined by your electronic mail system.

fax
An optional fax number for the role.

status
The availability of the role to participate in a workflow process. The possible statuses are ACTIVE, EXTLEAVE, INACTIVE, and TMPLEAVE. If null, the procedure sets the status to 'ACTIVE'.

expiration_date
The date at which the role is no longer valid in the directory service.

parent_orig_system
An optional code for the originating system of an entity that you want to mark as being related to this role.

parent_orig_system_id
The primary key that identifies the parent entity in the parent originating system.

owner_tag
A code to identify the program or application that owns the information for this role.

Related Topics

Setting Up an Oracle Workflow Directory Service, *Oracle Workflow Administrator's Guide*

AddUsersToAdHocRole

Syntax

```
procedure AddUsersToAdHocRole
  (role_name in varchar2,
   role_users in varchar2);
```

Description

Adds users to an existing ad hoc role.

> **Note:** A role can contain only individual users as its members. It cannot contain another role.

Arguments (input)

role_name
The internal name of the ad hoc role.

role_users
The list of users, delimited by spaces or commas. Users can be ad hoc users or users defined in an application, but they must already be defined in the Oracle Workflow directory service.

AddUsersToAdHocRole2

Syntax

```
procedure AddUsersToAdHocRole2
  (role_name in varchar2,
   role_users in WF_DIRECTORY.UserTable);
```

Description

Adds users to a existing ad hoc role. *AddUsersToAdHocRole2()* accepts the list of users in the `WF_DIRECTORY.UserTable` format, which lets you include user names that contain spaces or commas.

Note: A role can contain only individual users as its members. It cannot contain another role.

Arguments (input)

role_name
The internal name of the ad hoc role.

role_users
The list of users, as a table in the `WF_DIRECTORY.UserTable` format. Users can be ad hoc users or users defined in an application, but they must already be defined in the Oracle Workflow directory service.

RemoveUsersFromAdHocRole

Syntax

```
procedure RemoveUsersFromAdHocRole
  (role_name in varchar2,
   role_users in varchar2 default null);
```

Description

Removes users from an existing ad hoc role.

Arguments (input)

role_name
The internal name of the ad hoc role.

role_users
List of users to remove from the ad hoc role. The users are delimited by commas or spaces. If null, all users are removed from the role.

SetAdHocUserStatus

Syntax

```
procedure SetAdHocUserStatus
  (user_name in varchar2,
   status in varchar2 default 'ACTIVE');
```

Description

Sets the status of an ad hoc user as `'ACTIVE'` or `'INACTIVE'`.

Important: If you implement Oracle Internet Directory integration, you must maintain your users only through Oracle Internet Directory. You must not use the *SetAdHocUserStatus()* API to update user information in the WF_LOCAL_ROLES table, because you risk discrepancies in your user information and unpredictable results if you use any tool other than Oracle Internet Directory to maintain users after integrating with Oracle Internet Directory.

Arguments (input)

user_name
The internal name of the user.

status
A status of 'ACTIVE' or 'INACTIVE' to set for the user. If null, the status is 'ACTIVE'.

SetAdHocRoleStatus

Syntax

```
procedure SetAdHocRoleStatus
  (role_name in varchar2,
   status in varchar2 default 'ACTIVE');
```

Description

Sets the status of an ad hoc role as 'ACTIVE' or 'INACTIVE'.

Arguments (input)

role_name
The internal name of the role.

status
A status of 'ACTIVE' or 'INACTIVE' to set for the role. If null, the status is 'ACTIVE'.

SetAdHocUserExpiration

Syntax

```
procedure SetAdHocUserExpiration
  (user_name in varchar2,
   expiration_date in date default sysdate);
```

Description

Updates the expiration date for an ad hoc user.

Note that although users and roles whose expiration date has passed do not appear in the seeded WF_USERS, WF_ROLES, and WF_USER_ROLES views, they are not removed from the Workflow local tables until you purge them using *Directory()*. You should periodically purge expired users and roles in order to improve performance. See: Directory, page 2-82.

> **Important:** If you implement Oracle Internet Directory integration, you must maintain your users only through Oracle Internet Directory. You must not use the *SetAdHocUserExpiration()* API to update user information in the WF_LOCAL_ROLES table, because you risk discrepancies in your user information and unpredictable results if you use any tool other than Oracle Internet Directory to maintain users after integrating with Oracle Internet Directory.

Arguments (input)

user_name
The internal name of the ad hoc user.

expiration_date
New expiration date. If null, the procedure defaults the expiration date to sysdate.

SetAdHocRoleExpiration

Syntax

```
procedure SetAdHocRoleExpiration
  (role_name in varchar2,
   expiration_date in date default sysdate);
```

Description

Updates the expiration date for an ad hoc role.

Note that although users and roles whose expiration date has passed do not appear in the seeded WF_USERS, WF_ROLES, and WF_USER_ROLES views, they are not removed from the Workflow local tables until you purge them using *Directory()*. You should periodically purge expired users and roles in order to improve performance. See: Directory, page 2-82.

Arguments (input)

role_name
The internal name of the ad hoc role.

expiration_date
New expiration date. If null, the procedure defaults the expiration date to sysdate.

SetAdHocUserAttr

Syntax

```
procedure SetAdHocUserAttr
  (user_name in varchar2,
   display_name in varchar2 default null,
   notification_preference in varchar2 default null,
   language in varchar2 default null,
   territory in varchar2 default null,
   email_address in varchar2 default null,
   fax in varchar2 default null,
   parent_orig_system in varchar2 default null,
   parent_orig_system_id in number default null,
   owner_tag in varchar2 default null);
```

Description

Updates the attributes for an ad hoc user.

> **Important:** If you implement Oracle Internet Directory integration, you must maintain your users only through Oracle Internet Directory. You

must not use the *SetAdHocUserAttr()* API to update user information in the WF_LOCAL_ROLES table, because you risk discrepancies in your user information and unpredictable results if you use any tool other than Oracle Internet Directory to maintain users after integrating with Oracle Internet Directory.

Arguments (input)

user_name
The internal name of the ad hoc user to update.

display_name
A new display name for the ad hoc user. If null, the display name is not updated.

notification_preference
A new notification preference of 'MAILTEXT','MAILHTML','MAILATTH','MAILHTM2','QUERY','SUMMARY', or, for Oracle Applications only, 'SUMHTML'. If null, the notification preference is not updated.

language
A new value of the database NLS_LANGUAGE initialization parameter for the ad hoc user. If null, the language setting is not updated.

territory
A new value of the database NLS_TERRITORY initialization parameter for the ad hoc user. If null, the territory setting is not updated.

email_address
A new valid electronic mail address for the ad hoc user. If null, the electronic mail address is not updated.

fax
A new fax number for the ad hoc user. If null, the fax number is not updated.

parent_orig_system
An optional code for the originating system of an entity that you want to mark as being related to this user.

parent_orig_system_id
The primary key that identifies the parent entity in the parent originating system.

owner_tag
A code to identify the program or application that owns the information for this user.

SetAdHocRoleAttr

Syntax

```
procedure SetAdHocRoleAttr
  (role_name in varchar2,
   display_name in varchar2 default null,
   notification_preference in varchar2 default null,
   language in varchar2 default null,
   territory in varchar2 default null,
   email_address in varchar2 default null,
   fax in varchar2 default null,
   parent_orig_system in varchar2 default null,
   parent_orig_system_id in number default null,
   owner_tag in varchar2 default null);
```

Description

Updates the attributes for an ad hoc role.

Arguments (input)

role_name
The internal name of the ad hoc role to update.

display_name
A new display name for the ad hoc role. If null, the display name is not updated.

notification_preference
A new notification preference of 'MAILTEXT', 'MAILHTML', 'MAILATTH', 'MAILHTM2', 'QUERY', 'SUMMARY', or, for Oracle Applications only, 'SUMHTML'. If null, the notification preference is not updated.

language
A new value of the database NLS_LANGUAGE initialization parameter for the ad hoc role. If null, the language setting is not updated.

territory
A new value of the database NLS_TERRITORY initialization parameter for the ad hoc role. If null, the territory setting is not updated.

email_address
A new valid electronic mail address for the ad hoc role. If null, the electronic mail address is not updated.

fax
A new fax number for the ad hoc role. If null, the fax number is not updated.

parent_orig_system
An optional code for the originating system of an entity that you want to mark as being related to this role.

parent_orig_system_id
The primary key that identifies the parent entity in the parent originating system.

owner_tag
A code to identify the program or application that owns the information for this role.

ChangeLocalUserName

Syntax

```
function ChangeLocalUserName
  (OldName in varchar2,
   NewName in varchar2,
   Propagate in boolean default TRUE)
  return boolean;
```

Description

Changes a user's name in the WF_LOCAL_ROLES table. Returns TRUE if the name change completes successfully; otherwise, the API returns FALSE.

Arguments (input)

OldName
The current name of the user.

NewName
The new name for the user.

Propagate
Specify TRUE to change all occurrences of the old user name to the new user name.

IsMLSEnabled

Syntax

```
function IsMLSEnabled
  (p_orig_system in varchar2)
  return boolean;
```

Description

Determines whether Multilingual Support (MLS) is enabled for the specified originating system. Returns TRUE if MLS is enabled; otherwise the API returns FALSE.

Arguments (input)

p_orig_system
A system from which directory service information originates.

Workflow LDAP APIs

Call the following APIs to synchronize local user information in your Workflow directory service with the users in an LDAP directory such as Oracle Internet Directory. These APIs are defined in a PL/SQL package called WF_LDAP.

- Synch_changes, page 3-19
- Synch_all, page 3-19
- Schedule_changes, page 3-20

Related Topics

Synchronizing Workflow Directory Services with Oracle Internet Directory, *Oracle Workflow Administrator's Guide*

Synch_changes

Syntax

```
function synch_changes
  return boolean;
```

Description

Determines whether there have been any user changes to an LDAP directory since the last synchronization by querying the LDAP change log records; if there are any changes, including creation, modification, and deletion, *Synch_changes()* stores the user attribute information in an attribute cache and raises the `oracle.apps.global.user.change` event to alert interested parties. The function connects to the LDAP directory specified in the global workflow preferences. One event is raised for each changed user.

If the function completes successfully, it returns `TRUE`; otherwise, if it encounters an exception, it returns `FALSE`.

Related Topics

Synchronizing Workflow Directory Services with Oracle Internet Directory, *Oracle Workflow Administrator's Guide*

Setting Global User Preferences, *Oracle Workflow Administrator's Guide*

User Entry Has Changed Event, *Oracle Workflow Developer's Guide*

Synch_all

Syntax

```
function synch_all
  return boolean;
```

Description

Retrieves all users from an LDAP directory, stores the user attribute information in an attribute cache, and raises the `oracle.apps.global.user.change` event to alert interested parties. The function connects to the LDAP directory specified in the global workflow preferences. One event is raised for each user.

Because *Synch_all()* retrieves information for all users stored in the LDAP directory, you should use this function only once during setup, or as required for recovery or cleanup. Subsequently, you can use *Synch_changes()* or *Schedule_changes()* to retrieve only changed user information.

If the function completes successfully, it returns `TRUE`; otherwise, if it encounters an exception, it returns `FALSE`.

The standalone Oracle Workflow installation runs*Synch_all()* to begin your Workflow directory service synchronization with Oracle Internet Directory if you implement Oracle Internet Directory integration.

Related Topics

Synchronizing Workflow Directory Services with Oracle Internet Directory, *Oracle Workflow Administrator's Guide*

Setting Global User Preferences, *Oracle Workflow Administrator's Guide*

User Entry Has Changed Event, *Oracle Workflow Developer's Guide*

Synch_changes, page 3-19

Schedule_changes, page 3-20

Schedule_changes

Syntax

```
procedure schedule_changes
  (l_day in pls_integer default 0,
   l_hour in pls_integer default 0,
   l_minute in pls_integer default 10);
```

Description

Runs the *Synch_changes()* API repeatedly at the specified time interval to check for user changes in an LDAP directory and alert interested parties of any changes. The default interval is ten minutes. *Schedule_changes()* submits a database job using the DBMS_JOB utility to run *Synch_changes()*.

Run *Schedule_changes()* to maintain your Workflow directory service synchronization with Oracle Internet Directory if you implement Oracle Internet Directory integration.

Arguments (input)

l_day
The number of days in the interval to specify how often you want to run the *Synch_changes()* API. The default value is zero.

l_hour
The number of hours in the interval to specify how often you want to run the *Synch_changes()* API. The default value is zero.

l_minute
The number of minutes in the interval to specify how often you want to run the *Synch_changes()* API. The default value is ten.

Related Topics

Synchronizing Workflow Directory Services with Oracle Internet Directory, *Oracle Workflow Administrator's Guide*

Synch_changes, page 3-19

Workflow Local Synchronization APIs

The following APIs can be called in Oracle Applications to synchronize user and role information stored in application tables with the information in the Workflow local tables. These APIs are defined in a PL/SQL package called WF_LOCAL_SYNCH.

- Propagate_User, page 3-21
- Propagate_Role, page 3-25
- PropagateUserRole, page 3-29

> **Note:** The *Propagate_User_Role()* API from earlier versions of Oracle Workflow is replaced by the *PropagateUserRole()* API. The current version of Oracle Workflow still recognizes the *Propagate_User_Role()* API for backward compatibility, but moving forward, you should only use the new *PropagateUserRole()* API where appropriate.

Related Topics

Setting Up an Oracle Workflow Directory Service, *Oracle Workflow Administrator's Guide*

Propagate_User

Syntax

```
procedure Propagate_User
  (p_orig_system in varchar2,
   p_orig_system_id in number,
   p_attributes in wf_parameter_list_t,
   p_start_date in date default null,
   p_expiration_date in date default null);
```

Description

Synchronizes the information for a user from an application table with the WF_LOCAL_ROLES table and marks this record as an individual user by setting the user flag to `Y`. The user is identified by the specified originating system and originating system ID. The partition ID where the user's information is stored is set automatically depending on the originating system.

> **Note:** For Oracle Applications, only Oracle Applications users from the FND_USER table, Oracle Trading Community Architecture (TCA) person parties, and TCA contacts (relationship parties) should be synchronized using *Propagate_User()*. All other Oracle Applications modules should synchronize their information using *Propagate_Role()*.

The user information to be stored in the WF_LOCAL_ROLES table must be provided in the `WF_PARAMETER_LIST_T` format. You can use the *WF_EVENT.AddParameterToList()* API to add attributes to the list. The following table shows the attributes that should be included in the list to populate the required columns in WF_LOCAL_ROLES. The standard LDAP attribute names should be used for these attributes.

User Attributes

Database Column	Attribute Name
NAME	[USER_NAME]
DISPLAY_NAME	[DisplayName]
DESCRIPTION	[description]
NOTIFICATION_PREFERENCE	[orclWorkFlowNotificationPref]
LANGUAGE	[preferredLanguage]
TERRITORY	[orclNLSTerritory]
EMAIL_ADDRESS	[mail]
FAX	[FacsimileTelephoneNumber]
STATUS	[orclIsEnabled]
EXPIRATION_DATE	[ExpirationDate]
ORIG_SYSTEM	[orclWFOrigSystem]
ORIG_SYSTEM_ID	[orclWFOrigSystemID]
PARENT_ORIG_SYSTEM	[orclWFParentOrigSys]
PARENT_ORIG_SYSTEM_ID	[orclWFParentOrigSysID]
OWNER_TAG	[OWNER_TAG]
PERSON_PARTY_ID	[PERSON_PARTY_ID]
LAST_UPDATED_BY	[LAST_UPDATED_BY]
LAST_UPDATE_DATE	[LAST_UPDATE_DATE]
LAST_UPDATE_LOGIN	[LAST_UPDATE_LOGIN]
CREATED_BY	[CREATED_BY]
CREATION_DATE	[CREATION_DATE]

In normal operating mode, if any of these attributes except `USER_NAME` are not passed in the attribute list or are null, the existing value in the corresponding field in WF_LOCAL_ROLES remains the same. For example, if no e-mail address is passed, the existing e-mail address for the user is retained. However, you must always pass the `USER_NAME` attribute, because the *Propagate_User()* procedure uses this value in a `WHERE` condition and will fail if the `USER_NAME` is not provided. Also, if the user record does not already exist, you must pass all of the listed attributes since there are no existing values to use.

For more robust code, you should always pass all of the listed attributes when calling *Propagate_User()*. In this way you can avoid errors caused by trying to determine dynamically which attributes to pass.

> **Note:** If a display name is not provided in the attribute list when the user record is first created in normal operating mode, this

value is set by default to a composite value in the format *<orig_system>*:*<orig_system_ID>* in the user record in WF_LOCAL_ROLES. Additionally, if no notification preference is provided, the notification preference for the user record is set by default to MAILHTML, and if no status is provided, the status for the user record is set by default to ACTIVE. If no TCA person party ID is provided, Oracle Workflow uses a value consisting of the originating system and originating system ID as the person party ID.

You can also call *Propagate_User()* in overwrite mode by including a special attribute named WFSYNCH_OVERWRITE with a value of 'TRUE'. In overwrite mode, if one of the following attributes is not passed or is null, the procedure sets the value of the corresponding field in WF_LOCAL_ROLES to null, deleting the previous value.

- description
- preferredLanguage
- orclNLSTerritory
- mail
- FacsimileTelephoneNumber
- ExpirationDate
- orclWFParentOrigSys
- orclWFParentOrigSysID
- OWNER_TAG
- LAST_UPDATED_BY
- LAST_UPDATE_DATE
- LAST_UPDATE_LOGIN

Consequently, when you are using overwrite mode, you must pass values for all the attributes that you do not want to be null. Also, you must always pass the USER_NAME attribute.

Note: The DISPLAY_NAME, NOTIFICATION_PREFERENCE, STATUS, ORIG_SYSTEM, and ORIG_SYSTEM_ID columns in the WF_LOCAL_ROLES table have a NOT NULL constraint, so these columns retain their existing values if you do not pass a value for the corresponding attributes, even if you are using overwrite mode.

The NAME column in WF_LOCAL_ROLES also has a NOT NULL constraint, and you cannot omit the USER_NAME attribute in any case because it is required for the API.

Certain values, including the originating system, originating system ID, and expiration date, can be passed both as parameters for the *Propagate_User()* API and as attributes within the attribute list parameter. These values are repeated in the attribute list because *Propagate_User()* sends only the attribute list to the Entity Manager that coordinates LDAP integration, and not any of the procedure's other parameters.

- The originating system and originating system ID values that are passed as parameters to the procedure override any originating system and originating system ID values that are provided as attributes within the attribute list, if these values differ.
- Likewise, if an expiration date value is passed as a parameter to the procedure, that value overrides any expiration date value provided as an attribute within the attribute list. However, if the p_expiration_date parameter is null, the value of the ExpirationDate attribute will be used, if one is provided. You must provide the ExpirationDate attribute value in the following format:

  ```
  to_char(<your date variable>, WF_ENGINE.Date_Format)
  ```

Oracle Workflow also provides two additional special attributes that you can use to specify how the user information should be modified.

- DELETE - You can use this attribute when you want to remove a user from availability to participate in a workflow. If you include this attribute with a value of 'TRUE', the expiration date for the user in WF_LOCAL_ROLES is set to sysdate and the status is set to INACTIVE.

  > **Note:** If you also pass a value for the p_expiration_date parameter, however, that value will override the DELETE attribute. Additionally, if the p_expiration_date parameter is null but you include the ExpirationDate attribute, that attribute value will override the DELETE attribute. In these cases the user will remain valid and active until the specified expiration date.

- UpdateOnly - You can use this attribute for performance gain when you want to modify information for a user for whom a record already exists in WF_LOCAL_ROLES. If you include this attribute with a value of 'TRUE', the *Propagate_User()* API attempts to update the record directly, without first inserting the record.

  If this update attempt fails because a record does not already exist for that user, the procedure will then insert the record. However, the initial unsuccessful attempt will degrade performance, so you should only use the UpdateOnly attribute when you are certain that the user record already exists in WF_LOCAL_ROLES.

Arguments (input)

p_orig_system
A code that you assign to the directory repository that is the source of the user information.

p_orig_system_id
The primary key that identifies the user in this repository system.

p_attributes
A list of attribute name and value pairs containing information about the user.

p_start_date
The date at which the user becomes valid in the directory service.

p_expiration_date
The date at which the user is no longer valid in the directory service.

Related Topics

AddParameterToList, page 5-30

Propagate_Role

Syntax

```
procedure Propagate_Role
  (p_orig_system in varchar2,
   p_orig_system_id in number,
   p_attributes in wf_parameter_list_t,
   p_start_date in date default null,
   p_expiration_date in date default null);
```

Description

Synchronizes the information for a role from an application table with the WF_LOCAL_ROLES table and sets the user flag for the role to `N`. The role is identified by the specified originating system and originating system ID. The partition ID where the role's information is stored is set automatically depending on the originating system.

The role information to be stored in the WF_LOCAL_ROLES table must be provided in the `WF_PARAMETER_LIST_T` format. You can use the *WF_EVENT.AddParameterToList()* API to add attributes to the list. The following table shows the attributes that should be included in the list to populate the required columns in WF_LOCAL_ROLES. The standard LDAP attribute names should be used for these attributes.

Role Attributes

Database Column	Attribute Name
NAME	[USER_NAME]
DISPLAY_NAME	[DisplayName]
DESCRIPTION	[description]
NOTIFICATION_PREFERENCE	[orclWorkFlowNotificationPref]
LANGUAGE	[preferredLanguage]
TERRITORY	[orclNLSTerritory]
EMAIL_ADDRESS	[mail]
FAX	[FacsimileTelephoneNumber]
STATUS	[orclIsEnabled]
EXPIRATION_DATE	[ExpirationDate]
ORIG_SYSTEM	[orclWFOrigSystem]
ORIG_SYSTEM_ID	[orclWFOrigSystemID]
PARENT_ORIG_SYSTEM	[orclWFParentOrigSys]
PARENT_ORIG_SYSTEM_ID	[orclWFParentOrigSysID]
OWNER_TAG	[OWNER_TAG]
LAST_UPDATED_BY	[LAST_UPDATED_BY]
LAST_UPDATE_DATE	[LAST_UPDATE_DATE]
LAST_UPDATE_LOGIN	[LAST_UPDATE_LOGIN]
CREATED_BY	[CREATED_BY]
CREATION_DATE	[CREATION_DATE]

In normal operating mode, if any of these attributes except `USER_NAME` are not passed in the attribute list or are null, the existing value in the corresponding field in WF_LOCAL_ROLES remains the same. For example, if no e-mail address is passed, the existing e-mail address for the role is retained. However, you must always pass the `USER_NAME` attribute, because the *Propagate_Role()* procedure uses this value in a `WHERE` condition and will fail if the `USER_NAME` is not provided. Also, if the user record does not already exist, you must pass all of the listed attributes since there are no existing values to use.

For more robust code, you should always pass all of the listed attributes when calling *Propagate_Role()*. In this way you can avoid errors caused by trying to determine dynamically which attributes to pass.

> **Note:** If a display name is not provided in the attribute list when the role record is first created in normal operating mode, this value is set by default to a composite value in the format *<orig_system>*:*<orig_system_ID>* in the role record in

WF_LOCAL_ROLES. Additionally, if no notification preference is provided, the notification preference for the role record is set by default to MAILHTML, and if no status is provided, the status for the role record is set by default to ACTIVE.

You can also call *Propagate_Role()* in overwrite mode by including a special attribute named WFSYNCH_OVERWRITE with a value of 'TRUE'. In overwrite mode, if one of the following attributes is not passed or is null, the procedure sets the value of the corresponding field in WF_LOCAL_ROLES to null, deleting the previous value.

- description
- preferredLanguage
- orclNLSTerritory
- mail
- FacsimileTelephoneNumber
- ExpirationDate
- orclWFParentOrigSys
- orclWFParentOrigSysID
- OWNER_TAG
- LAST_UPDATED_BY
- LAST_UPDATE_DATE
- LAST_UPDATE_LOGIN

Consequently, when you are using overwrite mode, you must pass values for all the attributes that you do not want to be null. Also, you must always pass the USER_NAME attribute.

> **Note:** The DISPLAY_NAME, NOTIFICATION_PREFERENCE, STATUS, ORIG_SYSTEM, and ORIG_SYSTEM_ID columns in the WF_LOCAL_ROLES table have a NOT NULL constraint, so these columns retain their existing values if you do not pass a value for the corresponding attributes, even if you are using overwrite mode.
>
> The NAME column in WF_LOCAL_ROLES also has a NOT NULL constraint, and you cannot omit the USER_NAME attribute in any case because it is required for the API.

Certain values, including the originating system, originating system ID, and expiration date, can be passed both as parameters for the *Propagate_Role()* API and as attributes within the attribute list parameter. These values are repeated in the attribute list because *Propagate_Role()* sends only the attribute list to the Entity Manager that coordinates LDAP integration, and not any of the procedure's other parameters.

- The originating system and originating system ID values that are passed as parameters to the procedure override any originating system and originating system ID values that are provided as attributes within the attribute list, if these values differ.
- Likewise, if an expiration date value is passed as a parameter to the procedure, that value overrides any expiration date value provided as an attribute within the

attribute list. However, if the p_expiration_date parameter is null, the value of the ExpirationDate attribute will be used, if one is provided. You must provide the ExpirationDate attribute value in the following format:

```
to_char(<your date variable>, WF_ENGINE.Date_Format)
```

Oracle Workflow also provides two additional special attributes that you can use to specify how the role information should be modified.

- DELETE - You can use this attribute when you want to remove a role from availability to participate in a workflow. If you include this attribute with a value of 'TRUE', the expiration date for the role in WF_LOCAL_ROLES is set to sysdate and the status is set to INACTIVE.

  Note: If you also pass a value for the p_expiration_date parameter, however, that value will override the DELETE attribute. Additionally, if the p_expiration_date parameter is null but you include the ExpirationDate attribute, that attribute value will override the DELETE attribute. In these cases the role will remain valid and active until the specified expiration date.

- UpdateOnly - You can use this attribute for performance gain when you want to modify information for a role for which a record already exists in WF_LOCAL_ROLES. If you include this attribute with a value of 'TRUE', the *Propagate_Role()* API attempts to update the record directly, without first inserting the record.

  If this update attempt fails because a record does not already exist for that role, the procedure will then insert the record. However, the initial unsuccessful attempt will degrade performance, so you should only use the UpdateOnly attribute when you are certain that the role record already exists in WF_LOCAL_ROLES.

  Note: In Oracle Applications, if an Oracle Human Resources person role with an originating system of PER_ROLE is propagated using *Propagate_Role()*, and that person is linked to an Oracle Applications user, then the procedure updates the corresponding user record with an originating system of PER in WF_LOCAL_ROLES, as well as the person record.

Arguments (input)

p_orig_system
A code that you assign to the directory repository that is the source of the role information.

p_orig_system_id
The primary key that identifies the role in this repository system.

p_attributes
A list of attribute name and value pairs containing information about the role.

p_start_date
The date at which the role becomes valid in the directory service.

p_expiration_date
The date at which the role is no longer valid in the directory service.

Related Topics

AddParameterToList, page 5-30

PropagateUserRole

Syntax

```
procedure PropagateUserRole
  (p_user_name in varchar2,
   p_role_name in varchar2,
   p_user_orig_system in varchar2 default null,
   p_user_orig_system_id in number default null,
   p_role_orig_system in varchar2 default null,
   p_role_orig_system_id in number default null,
   p_start_date in date default null,
   p_expiration_date in date default null,
   p_overwrite in boolean default FALSE,
   p_raiseErrors in boolean default FALSE,
   p_parent_orig_system in varchar2 default null,
   p_parent_orig_system_id in varchar2 default null,
   p_ownerTag in varchar2 default null,
   p_createdBy in number default null,
   p_lastUpdatedBy in number default null,
   p_lastUpdateLogin in number default null,
   p_creationDate in date default null,
   p_lastUpdateDate in date default null);
```

Description

Synchronizes the information for an association of a user and a role from an application table with the WF_LOCAL_USER_ROLES table.

Arguments (input)

p_user_name
The internal name of the user.

p_role_name
The internal name of the role.

p_user_orig_system
A code that you assign to the directory repository that is the source of the user information.

p_user_orig_system_id
The primary key that identifies the user in this repository system.

p_role_orig_system
A code that you assign to the directory repository that is the source of the role information.

p_role_orig_system_id
The primary key that identifies the role in this repository system.

p_start_date
The date at which the association of this user with this role becomes valid in the directory service.

p_expiration_date
The date at which the association of this user with this role is no longer valid in the directory service.

p_overwrite
Specify `TRUE` or `FALSE` to determine whether to propagate the information in overwrite mode. In overwrite mode, if any attribute is not passed or is null, the procedure sets the value of the corresponding field in WF_LOCAL_USER_ROLES to null, deleting the previous value.

> **Note:** Overwrite mode does not affect the user name and role name attributes. You must pass values for these parameters, because they are required for this procedure, and because the `USER_NAME` and `ROLE_NAME` columns in the WF_LOCAL_USER_ROLES table have a `NOT NULL` constraint.

p_raiseErrors
Specify `TRUE` or `FALSE` to determine whether the procedure should raise an exception if it encounters an error.

p_parent_orig_system
A code for the originating system of an entity that you want to mark as being related to the association of this user with this role.

p_parent_orig_system_id
The primary key that identifies the parent entity in the parent originating system.

p_ownerTag
A code to identify the program or application that owns the information for the association of this user with this role.

p_createdBy
Standard Who column.

p_lastUpdatedBy
Standard Who column.

p_lastUpdateLogin
Standard Who column.

p_creationDate
Standard Who column.

p_lastUpdateDate
Standard Who column.

Workflow Role Hierarchy APIs

The following APIs can be called by an application program or a workflow function in the runtime phase to manage role hierarchy relationships in the Oracle E-Business Suite directory service. These APIs are defined in a PL/SQL package called WF_ROLE_HIERARCHY.

- AddRelationship, page 3-31
- ExpireRelationship, page 3-31
- GetRelationships, page 3-32

- GetAllRelationships, page 3-32

Related Topics

Setting Up an Oracle Workflow Directory Service, *Oracle Workflow Administrator's Guide*

AddRelationship

Syntax

```
function AddRelationship
  (p_sub_name in varchar2,
   p_super_name in varchar2,
   p_deferMode in boolean default FALSE,
   p_enabled in varchar2 default 'Y')
  return number;
```

Description

Creates a hierarchical relationship between two roles in the WF_ROLE_HIERARCHIES table and returns the relationship ID.

Arguments (input)

p_sub_name
The internal name of the subordinate role.

p_super_name
The internal name of the superior role.

p_deferMode
Specify `TRUE` or `FALSE` to determine whether to defer propagation of the new relationship. If you specify `FALSE`, existing user and role assignments are updated according to the new relationship, without deferral.

p_enabled
Specify `'Y'` if the relationship is initially enabled or `'N'` if the relationship is initially disabled.

ExpireRelationship

Syntax

```
function ExpireRelationship
  (p_sub_name in varchar2,
   p_super_name in varchar2,
   p_defer_mode in boolean default FALSE)
  return number;
```

Description

Expires a hierarchical relationship between two roles in the WF_ROLE_HIERARCHIES table and returns the relationship ID.

Arguments (input)

p_sub_name
The internal name of the subordinate role.

p_super_name
The internal name of the superior role.

p_defer_mode
Specify TRUE or FALSE to determine whether to defer propagation of the expired relationship. If you specify FALSE, existing user and role assignments are updated according to the expired relationship, without deferral.

GetRelationships

Syntax

```
procedure GetRelationships
  (p_name in varchar2,
   p_superiors out WF_ROLE_HIERARCHY.relTAB,
   p_subordinates out WF_ROLE_HIERARCHY.relTAB,
   p_direction in VARCHAR2 default 'BOTH');
```

Description

Retrieves the hierarchical relationships for the specified role and returns a table of superior roles and a table of subordinate roles. *GetRelationships()* stops retrieving relationships in a hierarchy when it encounters a disabled relationship.

Arguments (input)

p_name
The internal name of the role.

p_direction
Specify 'SUPERIORS' to retrieve superior roles for this role, 'SUBORDINATES' to retrieve subordinate roles for this role, or 'BOTH' to retrieve both superior and subordinate roles.

GetAllRelationships

Syntax

```
procedure GetAllRelationships
  (p_name in varchar2,
   p_superiors out WF_ROLE_HIERARCHY.relTAB,
   p_subordinates out WF_ROLE_HIERARCHY.relTAB,
   p_direction in VARCHAR2 default 'BOTH');
```

Description

Retrieves the hierarchical relationships for the specified role and returns a table of superior roles and a table of subordinate roles. *GetAllRelationships()* retrieves all hierarchical relationships, whether they are enabled or disabled.

Arguments (input)

p_name
The internal name of the role.

p_direction
Specify 'SUPERIORS' to retrieve superior roles for this role, 'SUBORDINATES' to retrieve subordinate roles for this role, or 'BOTH' to retrieve both superior and subordinate roles.

Workflow Preferences API

Call the following API to retrieve user preference information. The API is defined in the PL/SQL package called WF_PREF.

get_pref

Syntax

```
function get_pref
  (p_user_name in varchar2,
   p_preference_name in varchar2)
  return varchar2;
```

Description

Retrieves the value of the specified preference for the specified user.

Arguments (input)

p_user_name
The internal name of the user. To retrieve the value for a global preference, specify the user as -WF_DEFAULT-.

p_preference_name
The name of the user preference whose value you wish to retrieve. Valid preference names are:

- LANGUAGE
- TERRITORY
- MAILTYPE
- DMHOME
- DATEFORMAT

4

Notification System APIs

This chapter describes the APIs for the Oracle Workflow Notification System. The APIs consist of PL/SQL and Java functions and procedures that you can use to access the Notification System.

This chapter covers the following topics:

- Overview of the Oracle Workflow Notification System
- Notification APIs
- Notification Mailer Utility API

Overview of the Oracle Workflow Notification System

Oracle Workflow communicates with users by sending notifications. Notifications contain messages that may request users to take some type of action and/or provide users with information. You define the notification activity and the notification message that the notification activity sends in the Workflow Builder. The messages may have optional attributes that can specify additional resources and request responses.

Users can query their notifications online using the Notifications Web page in an HTML browser. Users can also receive notifications in their e-mail applications. E-mail notifications can contain HTML content or include other documents as optional attachments. The Notification System delivers the messages and processes the incoming responses.

Related Topics

Notification Model, page 4-1

Notification Document Type Definition, page 4-6

Notification APIs, page 4-14

Notification Mailer Utility API, page 4-44

Notification Model

A notification activity in a workflow process consists of a design-time message and a list of message attributes. In addition, there may be a number of runtime named values called item type attributes from which the message attributes draw their values.

The Workflow Engine moves through the workflow process, evaluating each activity in turn. Once it encounters a notification activity, the engine makes a call to the Notification System *Send()* or *SendGroup()* API to send the notification.

Sending Notification Messages

The *Send()* API or the *SendGroup()* API is called by the Workflow Engine when it encounters a notification activity. These APIs do the following:

- Check that the performer role of the notification activity is valid.
- Identify the notification preference for of the performer role.
- Look up the message attributes for the message.
  - If a message attribute is of source SEND, the *Send()* or *SendGroup()* API retrieves its value from the item type attribute that the message attribute references. If the procedure cannot find an item type attribute, it uses the default value of the message attribute, if available. The Subject and Body of the message may include message attributes of source SEND, which the *Send()* or *SendGroup()* API token replaces with each attribute's current value when creating the notification.
  - If a message includes a message attribute of source RESPOND, the *Send()* or *SendGroup()* API checks to see if it has a default value assigned to it. The procedure then uses these RESPOND attributes to create the default response section of the notification.
- 'Construct' the notification content by inserting relevant information into the Workflow notification tables.
- Update the notification activity's status to `'NOTIFIED'` if a response is required or to `'COMPLETE'` if no response is required.

  Note: If a notification activity sends a message that is for the performer's information only (FYI), where there are no RESPOND message attributes associated with it, the notification activity gets marked as complete as soon as the Notification System delivers the message.

  Note: In the case of a voting activity, the status is updated to `'WAITING'` instead of `'NOTIFIED'`. See: Special Handling of Voting Activities, page 4-5.

- Raise the `oracle.apps.wf.notification.send` event. When this event is processed, a notification mailer generates an e-mail version of the notification if the performer role of a notification has a notification preference of `MAILTEXT`, `MAILHTML`, `MAILHTM2`, or `MAILATTH`, and sends the e-mail to the performer. For roles with a notification preference of `SUMMARY`, or, for Oracle Applications only, `SUMHTML`, a summary e-mail is sent when the `oracle.apps.wf.notification.summary.send` event is raised. See: Implementing Notification Mailers, *Oracle Workflow Administrator's Guide*.

Users who view their notifications from the Notifications Web page, regardless of their notification preferences, are simply querying the Workflow notification tables from this interface.

A notification recipient can perform the following actions with the notification:

- Respond to the notification or close the notification if it does not require a response. See: Processing a Notification Response, page 4-3.
- Forward the notification to another role. See: Forwarding a Notification, page 4-3.
- Transfer ownership of the notification to another role. See: Transferring a Notification, page 4-4.
- Request more information about the notification from another role, or respond to such a request with more information. See: Requesting More Information About a Notification, page 4-4.
- Ignore the notification and let it time out. See: Processing a Timed Out Notification, page 4-5.

> **Note:** In Oracle Applications, you can use the WF: Notification Reassign Mode profile option to determine whether users can reassign notifications by forwarding (also known as delegating) the notifications, transferring the notifications, or both. See: Setting the WF: Notification Reassign Mode, *Oracle Workflow Administrator's Guide*.

Processing a Notification Response

After a recipient responds, the Notification Details Web page or a notification mailer assigns the response values to the notification response attributes and calls the notification *Respond()* API. The *Respond()* API first calls a notification callback function to execute the notification activity's post-notification function (if it has one) in `VALIDATE` mode. In this mode, the post-notification function can validate the response values before accepting and recording the response. For example, if the notification requires an electronic signature, the post-notification function can run in `VALIDATE` mode to verify the response values and inform the user of any errors before requiring the user to enter a signature. If the post-notification function raises an exception, the response is aborted. See: Post-notification Functions, page 2-9.

Next, *Respond()* calls the notification callback function to execute the post-notification function in `RESPOND` mode. The post-notification function may interpret the response and perform tightly-coupled post-response processing. Again, if the post-notification function raises an exception, the response is aborted.

If no exception is raised, *Respond()* marks the notification as closed and then calls the notification callback function again in `SET` mode to update the corresponding item attributes with the `RESPOND` notification attributes values. If the notification message prompts for a response that is specified in the Result tab of the message's property page, that response value is also set as the result of the notification activity.

Finally, *Respond()* calls *WF_ENGINE.CompleteActivity()* to inform the engine that the notification activity is complete so it can transition to the next qualified activity.

Forwarding a Notification

If a recipient forwards a notification to another role, the Notification Details Web page calls the Notification System's *Forward()* API.

> **Note:** The Notification System is not able to track notifications that are forwarded via e-mail. It records only the eventual responder's e-mail address and any Respond message attributes values included in the response.

The *Forward()* API validates the role, then calls a notification callback function to execute the notification activity's post-notification function (if it has one) in `FORWARD` mode. As an example, the post-notification function may verify whether the role that the notification is being forwarded to has appropriate authority to view and respond to the notification. If it doesn't, the post-notification function may return an error and prevent the Forward operation from proceeding. See: Post-notification Functions, page 2-9.

Forward() then forwards the notification to the new role, along with any appended comments.

> **Note:** *Forward()* does not update the owner or original recipient of the notification.

Transferring a Notification

If a recipient transfers the ownership of a notification to another role, the Notification Details Web page calls the Notification System's *Transfer()* API.

> **Note:** Recipients who view notifications from an e-mail application cannot transfer notifications. To transfer a notification, the recipient must use the Notifications Web page.

The *Transfer()* API validates the role, then calls a notification callback function to execute the notification activity's post-notification function (if it has one) in `TRANSFER` mode. As an example, the post-notification function may verify whether the role that the notification is being transferred to has appropriate authority. If it doesn't, the post-notification function may return an error and prevent the Transfer operation from proceeding. See: Post-notification Functions, page 2-9.

Transfer() then assigns ownership of the notification to the new role, passing along any appended comments. Note that a transfer is also recorded in the comments of the notification.

Requesting More Information About a Notification

If a recipient requests more information about the notification from another role, the Notification Details Web page calls the Notification System's *UpdateInfo()* API, or a notification mailer calls the Notification System's *UpdateInfo2()* API.

The *UpdateInfo()* or *UpdateInfo2()* API calls a notification callback function to execute the notification activity's post-notification function (if it has one) in `QUESTION` mode. As an example, the post-notification function may verify that the request is directed to a role that has appropriate authority to view the notification. If it doesn't, the post-notification function may return an error and prevent the request for more information from being sent. See: Post-notification Functions, page 2-9.

If no error is returned, the API then sends the request for more information to the designated role. Note that a request for information is also recorded in the comments of the notification.

If the recipient of a request for more information responds with answering information, the Notification Details Web page calls the Notification System's *UpdateInfo()* API if the responder is logged in individually or the *UpdateInfoGuest()* API if the responder is logged in as the `GUEST` user, or a notification mailer calls the Notification System's *UpdateInfo2()* API.

The *UpdateInfo()*,*UpdateInfoGuest()*, or *UpdateInfo2()* API calls a notification callback function to execute the notification activity's post-notification function (if it has one) in ANSWER mode. As an example, the post-notification function may validate the answering information. If such validation fails, the post-notification function may return an error and prevent the answer from being sent. See: Post-notification Functions, page 2-9.

If no error is returned, the API then sends the answering information back to the recipient role of the original notification. Note that an answer to a request for information is also recorded in the comments of the notification.

Processing a Timed Out Notification

Timed out notification or subprocess activities are initially detected by the background engine. Background engines set up to handle timed out activities periodically check for activities that have time out values specified. If an activity does have a time out value, and the current date and time exceeds that time out value, the background engine marks that activity's status as 'TIMEOUT' and calls the Workflow Engine. The Workflow Engine then resumes by trying to execute the activity to which the <Timeout> transition points.

Special Handling of Voting Activities

A voting activity by definition is a notification activity that:

- Has its roles expanded, so that an individual copy of the notification message is sent to each member of the Performer role.
- Has a message with a specified Result, that requires recipients to respond from a list of values.
- Has a post-notification function associated with it that contains logic in the RUN mode to process the polled responses from the Performer members to generate a single response that the Workflow Engine interprets as the result of the notification activity. See: Voting Activity, *Oracle Workflow Developer's Guide*.

Once the Notification System sends the notification for a voting activity, it marks the voting activity's status as 'NOTIFIED'. The voting activity's status is updated to 'WAITING' as soon as some responses are received, but not enough responses are received to satisfy the voting criteria.

The individual role members that each receive a copy of the notification message can then respond or forward the notification, or request or respond with more information, if they use e-mail or the Worklist Web pages to access the notification. They can also transfer the notification if they use the Worklist Web pages.

The notification user interface calls the appropriate *Respond()*, *Forward()*, *Transfer()*, *UpdateInfo()*, *UpdateInfo2()*, or *UpdateInfoGuest()* API, depending on the action that the performer takes. Each API in turn calls the notification callback function to execute the post-notification function in VALIDATE and RESPOND, FORWARD, TRANSFER, QUESTION, or ANSWER mode, as appropriate. When the Notification System finishes executing the post-notification function in FORWARD or TRANSFER mode, it carries out the Forward or Transfer operation, respectively. When the Notification System finishes executing the post-notification function in QUESTION or ANSWER mode, it sends the request for more information to the designated role or the answer to the requesting role, respectively.

When the Notification System completes execution of the post-notification function in RESPOND mode, the Workflow Engine then runs the post-notification function again in

RUN mode. It calls the function in RUN mode after all responses are received to execute the vote tallying logic.

Also if the voting activity is reset to be reexecuted as part of a loop, or if it times out, the Workflow Engine runs the post-notification function in CANCEL or TIMEOUT mode, respectively. The logic for TIMEOUT mode in a voting activity's post-notification function should identify how to tally the votes received up until the timeout.

Notification Document Type Definition

The following document type definition (DTD) describes the required structure for the XML document that represents a notification. The Notification System uses this structure to communicate messages to a notification mailer. The following table shows the level, tag name, and description for each element in the DTD.

Notification DTD

Level	Tag	Description
1	<NOTIFICATIONGROUP maxcount="">	The <NOTIFICATIONGROUP> tag is the opening tag for the XML structure. The maxcount attribute defines the maximum number of <NOTIFICATION> tags to expect. This number may not be reached, but will not be exceeded within the <NOTIFICATIONGROUP> tag.

Level	Tag	Description
2	<NOTIFICATION nid="" language="" territory= "" codeset="" priority="" accesskey="" node="" item_ type="" message_name="" nidstr="">	The <NOTIFICATION> element defines a single message entity. A <NOTIF ICATION> is a repeating structure within <NOTIFICAT IONGROUP>, the number of which will not exceed the specified maxcount value. Each <NOTIFICATION> element for a notification sent by the Notification System is identified by its unique nid attribute, which is the notification ID. For messages received from an external source, such as notification responses from users, the notification ID should be zero (0). The language and territory values represent the language and territory preferences of the notification recipients. The codeset attribute is the preferred codeset associated with the language in the WF_LANGUAGES table. The value of the codeset attribute must be in the Oracle Database codeset notation. If the Reset NLS parameter is selected for the mailer that sends this notification, then the e-mail will be encoded to the IANA (Internet Assigned Numbers Authority) equivalent of the Oracle Database codeset. The priority attribute is the relative priority for the message. A priority of 1 through 33 is high, 34 through 66 is normal, and 67 through 99 is low. The accesskey and node attributes store information for inbound response messages. These attributes are used together with the nid attribute to validate the response. The item_type attribute is the internal name of the item type that owns the notification. The message_name attribute is the internal message name for the notification within that item type. These two attributes are provided for reference and are not used by a notification mailer. The nidstr attribute is for internal use only.

Level	Tag	Description
3	<HEADER>	<The HEADER> element defines the envelope information for the message, which contains the details of the recipients, where the message was sent from, and the subject for the message.
4	<RECIPIENTLIST>	The <RECIPIENTLIST> tag enables the message to be sent to more than one recipient. The first recipient in the list is treated as the primary recipient. Subsequent recipients will receive copies of the message. All recipients in the list will receive the same e-mail in the language and formatting of the primary recipient's preferences.
5	<RECIPIENT name="" type="">	The <RECIPIENT> tag defines a recipient for the message. A <RECIPIENT> is a repeating structure within the <RECI PIENTLIST>. Each <RECIP IENT> is identified by its name attribute, which is the internal name of the recipient role. The type attribute contains the copy type for the recipient. Valid values for this attribute are `to`, `cc`, and `bcc`. If the type attribute is not provided, then the recipient is treated as having a copy type of `to`.
6	<NAME> </NAME>	The <NAME> tag defines the display name of the recipient.
6	<ADDRESS> </ADDRESS>	The <ADDRESS> tag defines the e-mail address of the recipient.
5	</RECIPIENT>	This tag marks the end of a <RECIPIENT> element.
4	</RECIPIENTLIST>	This tag marks the end of the <RECIPIENTLIST> element.

Level	Tag	Description
4	<FROM>	The <FROM> tag shows the sender of the message. For outbound notifications, the from role can be set using the #FROM_ROLE message attribute. The from role is also set to the role who reassigned the notification if this notification has been reassigned, to the requesting role if this notification is a request for more information, or to the responding role if this notification is a response to a request for more information. For inbound notifications, this information is determined by the From address of the incoming e-mail message.
5	<NAME> </NAME>	The <NAME> tag defines the display name of the sender.
5	<ADDRESS> </ADDRESS>	The <ADDRESS> tag defines the e-mail address of the sender.
4	</FROM>	This tag marks the end of the <FROM> element.
4	<SUBJECT> </SUBJECT>	The <SUBJECT> element holds the subject line of the notification.
3	</HEADER>	This tag marks the end of the <HEADER> element.
3	<CONTENT content-type="">	The <CONTENT> element holds the contents of the notification message. The <CONTENT> element contains one or more <BODYPART> elements. The content-type attribute contains the valid MIME type definition for the content within the <CONTENT> element. Valid values for the content-type attribute include `multipart/mixed`, `text/plain` and `text/html`. The first <BODY PART> element within the <CONTENT> tag is treated as the main content of the message, and will be the first component within a `multipart/*` message structure. Subsequent <BODY PART> elements are treated as attachments to the message.

Level	Tag	Description
4	<BODYPART content-type="">	The <BODYPART> tag represents a MIME component of the final message. This element contains a <MESSAGE> tag and optionally one or more <RESOURCE> tags. If the <RESOURCE> tags are implemented, then the content-type attribute must be defined for the <BODYPART> tag to explain the relationship of the <RESOURCE> elements to the <MESSAGE> element. The only valid value for this content-type attribute is `multipart/related`. The first <BODYPART> element is treated as the main content of the message. This content will be either `text/*` or `multipart/related`. The subsequent <BODY PART> elements contain any attachments as required by the notification message definition and the recipient's notification preference. Attachments may include an HTML-formatted version of the notification, a Notification Detail Link, and any message attributes for which the Attach Content option is selected. For inbound messages, the <BODYPART> element contains the message and any attachments where appropriate.

Level	Tag	Description
5	<MESSAGE content-type="" content-transfer-encoding="" content-disposition="" src="">	The content-type attribute contains the media type definition for the <MESSAGE> element. Valid values for this content-type attribute are `text/plain`, `text/html`, `multipart/mixed`, or `multipart/related`. The content-transfer-encoding attribute is an optional attribute to qualify further the encoding of the `text/plain` or `text/html` content. The content-disposition attribute specifies that the component is an attachment. The src attribute can optionally be defined if the content for the <MESSAGE> element is not readily available when the notification XML document is generated. The value of the src attribute must be a URL from which the content can be obtained during final e-mail message rendering.
-	<![CDATA[ ]]>	This structure holds the raw message content. If the content of a <RESOURCE> element should be merged into the content of the <MESSAGE> element, then the message content must include a token prefixed by an ampersand (`&`) to mark the position at which the resource content should appear. The token must match the token attribute value of the corresponding <RESOURCE> element.
5	</MESSAGE>	This tag marks the end of a <MESSAGE> element.

Level	Tag	Description
5	<RESOURCE content-type="" content-transfer-encoding="" content-disposition="" content-id="" src="" language="" territory="" page-type="" token="">	The content-type attribute contains the media type definition for the <RESOURCE> element. This value should be a media-type/subtype definition. The content-transfer-encoding attribute is an optional attribute to qualify further the encoding of the `text/plain` or `text/html` content. The content-disposition attribute specifies that the component is an attachment. The content-id attribute holds the unique content identifier for the component. This identifier is referenced within the content of the <MESSAGE> element. The src attribute can optionally be defined if the content for the <RESOURCE> element is not readily available when the notification XML document is generated. The value of the src attribute must be a URL from which the content can be obtained during final e-mail message rendering. In Oracle Applications only, if the src attribute is defined to refer to Oracle Applications Framework content and the message recipient is not an Oracle Applications user defined in the FND_USER table, then the language and territory attributes hold the language and territory preferences of the recipient. Also, if the src attribute refers to Oracle Applications Framework content, then the page-type attribute is set to the value `fwk` to identify Oracle Applications Framework as the source of the content. The page-type attribute should be defined only if the src attribute is defined correspondingly. The token attribute holds the token value used to mark the position at which the content of the <RESOURCE> element will be merged into the content of the <MESSAGE> element. Within the <MESSAGE> element, the token value is prefixed by an ampersand (`&`).

Level	Tag	Description
-	<![CDATA[ ]]>	This structure holds the content for the <RESOURCE> element.
5	</RESOURCE>	This tag marks the end of a <RESOURCE> element.
4	</BODYPART>	This tag marks the end of a <BODYPART> element.
3	</CONTENT>	This tag marks the end of the <CONTENT> element.
3	<RESPONSE>	The <RESPONSE> tag is implemented only for inbound notifications. It is not part of the specification for outbound notifications. The <RES PONSE> element contains one or more <ATTRIBUTE> elements, which hold the response values found in the incoming e-mail message. There should be an <ATTR IBUTE> tag for each response attribute associated with the notfication. However, only the RESULT message attribute is mandatory. The other respond attributes are optional. If no value is specified for a respond attribute, Oracle Workflow uses the default value defined for the message attribute.

Level	Tag	Description
4	<ATTRIBUTE name="" type="" format="">	The <ATTRIBUTE> tag holds the response value found in the incoming e-mail message for a particular response attribute. An <ATTRIBUTE> is a repeating structure within the <RESPONSE>. The name attribute for this element is the internal name of the response attribute. The type attribute of this element is the Oracle Workflow data type of the reponse attribute, which can be either TEXT, NUMBER, DATE, DOCUMENT, or LOOKUP. The format attribute for this element contains the format string for the response attribute. For response attributes of type LOOKUP, the format is used to identify the lookup type code according to the value of the name attribute. For other data types, the format attribute is not used.
-	<![CDATA]]>	This structure holds the response information to be assigned to the attribute.
4	</ATTRIBUTE>	This tag marks the end of an <ATTRIBUTE> element.
3	</RESPONSE>	This tag marks the end of a <RESPONSE> element.
2	</NOTIFICATION>	This tag marks the end of a <NOTIFICATION> element.
1	</NOTIFICATIONGROUP>	This tag marks the end of the <NOTIFICATIONGROUP> element.

Notification APIs

The following APIs can be called by a notification agent to manage notifications for a notification activity. The APIs are stored in the PL./SQL package called WF_NOTIFICATION.

Many of these notification APIs also have corresponding Java methods that you can call from any Java program to integrate with Oracle Workflow. The following list indicates whether the notification APIs are available as PL/SQL functions/procedures, as Java methods, or both. See: Oracle Workflow Java Interface, page 2-3.

Note: Java is case-sensitive and all Java method names begin with a lower case letter to follow Java naming conventions.

Note: The Notification System raises business events when a notification is sent, closed, canceled, or reassigned, or when a user responds to a notification. Although Oracle Workflow does not include any

predefined subscriptions to some of these events, you can optionally define your own subscriptions to these events if you want to perform custom processing when they occur. See: Notification Events, *Oracle Workflow Developer's Guide* and To Define an Event Subscription (for standalone Oracle Workflow), *Oracle Workflow Developer's Guide* or To Create or Update an Event Subscription (for Oracle Applications), *Oracle Workflow Developer's Guide*.

Send

PL/SQL Syntax

```
function SEND
 (role in varchar2,
  msg_type in varchar2,
  msg_name in varchar2,
  due_date in date default null,
  callback in varchar2 default null,
  context in varchar2 default null,
  send_comment in varchar2 default null
  priority in number default null)
 return number;
```

Java Syntax

```
public static BigDecimal send
  (WFContext wCtx,
   String role,
   String messageType,
   String messageName,
   String dueDate,
   String callback,
   String context,
   string sendComment,
   BigDecimal priority)
```

Description

This function sends the specified message to a role, returning a notification ID if successful. The notification ID must be used in all future references to the notification.

If your message has message attributes, the procedure looks up the values of the attributes from the message attribute table or it can use an optionally supplied callback interface function to get the value from the item type attributes table. A callback function can also be used when a notification is responded to.

> **Note:** If you are using the Oracle Workflow Notification System and its e-mail-based or Web-based notification client, the *Send* procedure implicitly calls the *WF_ENGINE.CB* callback function. If you are using your own custom notification system that does not call the Workflow Engine, then you must define your own callback function following a standard format and specify its name for the callback argument. See: Custom Callback Function, page 4-17.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

role
The role name assigned as the performer of the notification activity.

msg_type or messageType
The item type associated with the message.

msg_name or messageName
The message internal name.

due_date or dueDate
The date that a response is required. This optional due date is only for the recipient's information; it has no effect on processing.

callback
The callback function name used for communication of SEND and RESPOND source message attributes.

context
Context information passed to the callback function.

send_comment or sendComment
A comment presented with the message.

priority
The priority of the message, as derived from the `#PRIORITY` notification activity attribute. If `#PRIORITY` does not exist or if the value is null, the Workflow Engine uses the default priority of the message.

Custom Callback Function

A default callback function can be called at various points by the actions of the WF_NOTIFICATION APIs. You may provide your own custom callback function, but it must follow standard specifications.

If you do not need to handle attributes of type event through your callback function, the procedure must use the following standard API:

```
procedure <name in callback argument>
  (command in varchar2,
   context in varchar2,
   attr_name in varchar2,
   attr_type in varchar2,
   text_value in out varchar2,
   number_value in out number,
   date_value in out date);
```

If the callback function does need to handle attributes of type event, you can overload the procedure name with a second implementation that includes an additional argument for the event value. In this case you should also retain the original implementation for backward compatibility. However, it is recommended that you do not overload the procedure unless you have a requirement to handle event attributes.

The implementation of the procedure for event values must use the following standard API:

```
procedure <name in callback argument>
  (command in varchar2,
   context in varchar2,
   attr_name in varchar2,
   attr_type in varchar2,
   text_value in out varchar2,
   number_value in out number,
   date_value in out date
   event_value in out nocopy wf_event_t);
```

For ease of maintenance, you can define the procedure that does not include the `event_value` argument to call the procedure that does include that argument, so that you only need to maintain one version of your code. The following example shows one way to implement such a call:

Example

```
procedure your_callback
  (command in varchar2,
   context in varchar2,
   attr_name in varchar2,
   attr_type in varchar2,
   text_value in out varchar2,
   number_value in out number,
   date_value in out date)

is
  event_value wf_event_t;

begin
  your_package.your_callback(command, context, attr_name,
                             attr_type, text_value,
                             number_value, date_value,
                             event_value);

exception
  when others then
    Wf_Core.Context('your_package', 'your_callback',
                    command, context, attr_name, attr_type,
                    ':'||text_value||':'||to_char(number_value)
                    ||':'||to_char(date_value)||':');
    raise;

end your_callback;
```

Arguments (input)

command

Specify `GET`, `SET`, `COMPLETE`, `ERROR`, `TESTCTX`, `FORWARD`, `TRANSFER`, `QUESTION`, `ANSWER`, `VALIDATE`, or `RESPOND` as the action requested. Use `GET` to get the value of an attribute, `SET` to set the value of an attribute, `COMPLETE` to indicate that the response is complete, `ERROR` to set the associated notification activity to a status of `'ERROR'`, `TESTCTX` to test the current context by calling the item type's Selector/Callback function, or `FORWARD`, `TRANSFER`, `QUESTION`, `ANSWER`, `VALIDATE`, or `RESPOND` to execute the post-notification function in those modes.

context
The context passed to *SEND()* or *SendGroup()*. The format is *<itemtype>*:*<itemkey>*:*<activityid>*.

attr_name
An attribute name to set/get if command is `SET` or `GET`.

attr_type
An attribute type if command is `SET` or `GET`.

text_value
Value of a text attribute if command is `SET` or value of text attribute returned if command is `GET`.

number_value
Value of a number attribute if command is `SET` or value of a number attribute returned if command is `GET`.

date_value
Value of a date attribute if command is `SET` or value of a date attribute returned if command is `GET`.

event_value
Value of an event attribute if command is `SET` or value of an event attribute returned if command is `GET`. Required only if the procedure name is overloaded with a second implementation that handles event attributes.

> **Note:** The arguments `text_value`, `number_value`, and `date_value`, as well as `event_value` if you are using this argument, are mutually exclusive. That is, you should use only one of these arguments, depending on the value of the `attr_type` argument.

Examples

Example 1
When a notification is sent, the system calls the specified callback function once for each SEND attribute (to get the attribute value).

For each SEND attribute, call:

```
your_callback('GET', context, 'BUGNO', 'NUMBER', textval,
              numval, dateval);
```

Example 2
When the user responds to the notification, the callback is called again, once for each RESPOND attribute.

```
your_callback('SET', context, 'STATUS', 'TEXT',
              'COMPLETE', numval, dateval);
```

Example 3
Then finally the Notification System calls the `'COMPLETE'` command to indicate the response is complete.

```
your_callback('COMPLETE', context, attrname, attrtype,
              textval, numval, dateval);
```

Example 4
For a SEND attribute of type event, call the implementation that includes the `event_value` argument.

```
your_callback('GET', context, 'RECEIVE_EVENT', 'EVENT',
              textval, numval, dateval, eventval);
```

SendGroup

PL/SQL Syntax

```
function SendGroup
  (role in varchar2,
   msg_type in varchar2,
   msg_name in varchar2,
   due_date in date default null,
   callback in varchar2 default null,
   context in varchar2 default null,
   send_comment in varchar2 default null
   priority in number default null)
  return number;
```

Description

This function sends a separate notification to all the users assigned to a specific role and returns a number called a notification group ID, if successful. The notification group ID identifies that group of users and the notification they each received.

If your message has message attributes, the procedure looks up the values of the attributes from the message attribute table or it can use an optionally supplied callback interface function to get the value from the item type attributes table. A callback function can also be used when a notification is responded to.

> **Note:** If you are using the Oracle Workflow Notification System and its e-mail-based or Web-based notification client, the *Send* procedure implicitly calls the *WF_ENGINE.CB* callback function. If you are using your own custom notification system, then you must define your own callback function following a standard format and specify its name for the callback argument. See: Custom Callback Function, page 4-17.

Generally, this function is called only if a notification activity has 'Expanded Roles' checked in its properties page. If Expanded Roles is not checked, then the *Send()* function is called instead. See: Voting Activity, *Oracle Workflow Developer's Guide*.

Arguments (input)

role
The role name assigned as the performer of the notification activity.

msg_type
The item type associated with the message.

msg_name
The message internal name.

due_date
The date that a response is required. This optional due date is only for the recipient's information; it has no effect on processing.

callback
The callback function name used for communication of SEND source message attributes.

context
Context information passed to the callback function.

send_comment
A comment presented with the message.

priority
The priority of the message, as derived from the `#PRIORITY` notification activity attribute. If `#PRIORITY` does not exist or if the value is null, the Workflow Engine uses the default priority of the message.

Forward

PL/SQL Syntax

```
procedure FORWARD
  (nid in number,
   new_role in varchar2,
   forward_comment in varchar2 default null);
```

Java Syntax

```
public static boolean forward
  (WFContext wCtx,
   BigDecimal nid,
   String newRole
   String comment)
```

Description

This procedure delegates a notification to a new role to perform work, even though the original role recipient still maintains ownership of the notification activity. Also implicitly calls the Callback function specified in the Send or SendGroup function with `FORWARD` mode. A comment can be supplied to explain why the forward is taking place. Existing notification attributes (including due date) are not refreshed or otherwise changed. The Delegate feature in the Notification System calls this procedure. Note that when you forward a notification, the forward is recorded in the `USER_COMMENT` field of the notification.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

new_role or newRole
The role name of the person the note is reassigned to.

forward_comment or comment
An optional forwarding comment.

Example

Example

The following code excerpt shows an example of how to call *forward()* in a Java program. The example code is from the `WFTest.java` program.

```
// forward to MBEECH
System.out.println("Delegate Test");
count = WFNotificationAPI.workCount(ctx, "MBEECH");
System.out.println("There are " + count +
    " open notification(s) for" + " MBEECH");
System.out.println("Delegate nid " + myNid +
    " from BLEWIS to MBEECH");
WFNotificationAPI.forward(ctx, myNid, "MBEECH",
    "Matt, Please handle.");
count = WFNotificationAPI.workCount(ctx, "MBEECH");
System.out.println("There are " + count +
    " open notification(s) for" +
    " MBEECH after Delegate.");
```

Transfer

PL/SQL Syntax

```
procedure TRANSFER
  (nid in number,
   new_role in varchar2,
   forward_comment in varchar2 default null);
```

Java Syntax

```
public static boolean transfer
  (WFContext wCtx,
   BigDecimal nid,
   String newRole
   String comment)
```

Description

This procedure forwards a notification to a new role and transfers ownership of the notification to the new role. It also implicitly calls the Callback function specified in the Send or SendGroup function with `TRANSFER` mode. A comment can be supplied to explain why the forward is taking place. The Transfer feature in the Notification System calls this procedure. Note that when you transfer a notification, the transfer is recorded in the `USER_COMMENT` field of the notification.

> **Important:** Existing notification attributes (including due date) are not refreshed or otherwise changed except for `ORIGINAL_RECIPIENT`, which identifies the owner of the notification.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

new_role or newRole
The role name of the person the note is transferred to.

forward_comment or comment
An optional comment to append to notification.

Example

Example
The following code excerpt shows an example of how to call *transfer()* in a Java program. The example code is from the `WFTest.java` program.

```
// transfer to MBEECH
System.out.println("Transfer Test");
System.out.println("Transfer nid " + myNid +
    " from BLEWIS to MBEECH");
WFNotificationAPI.transfer(ctx, myNid, "MBEECH",
    "Matt, You own it now.");
count = WFNotificationAPI.workCount(ctx, "MBEECH");
System.out.println("There are " + count +
    " open notification(s) for" +
    " MBEECH after Transfer.");
```

Cancel

PL/SQL Syntax

```
procedure CANCEL
  (nid in number,
   cancel_comment in varchar2 default null);
```

Java Syntax

```
public static boolean cancel
    (WFContext wCtx,
     BigDecimal nid,
     String comment)
```

Description

This procedure may be invoked by the sender or administrator to cancel a notification. The notification status is then changed to `'CANCELED'` but the row is not removed from the WF_NOTIFICATIONS table until a purge operation is performed.

If the notification was delivered via e-mail and expects a response, a 'Canceled' e-mail is sent to the original recipient as a warning that the notification is no longer valid.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

cancel_comment or comment
An optional comment on the cancellation.

CancelGroup

PL/SQL Syntax

```
procedure CancelGroup
  (gid in number,
   cancel_comment in varchar2 default null);
```

Description

This procedure may be invoked by the sender or administrator to cancel the individual copies of a specific notification sent to all users in a notification group. The notifications are identified by the notification group ID (gid). The notification status is then changed to 'CANCELED' but the rows are not removed from the WF_NOTIFICATIONS table until a purge operation is performed.

If the notification was delivered via e-mail and expects a response, a 'Canceled' e-mail is sent to the original recipient as a warning that the notification is no longer valid.

Generally, this function is called only if a notification activity has 'Expanded Roles' checked in its properties page. If Expanded Roles is not checked, then the *Cancel()* function is called instead. See: Voting Activity, *Oracle Workflow Developer's Guide*.

Arguments (input)

gid
The notification group ID.

cancel_comment
An optional comment on the cancellation.

Respond

PL/SQL Syntax

```
procedure  RESPOND
  (nid in number,
   respond_comment in varchar2 default null,
   responder in varchar2 default null);
```

Java Syntax

```
public static boolean respond
  (WFContext wCtx,
   BigDecimal nid,
   String comment,
   String responder)
```

Description

This procedure may be invoked by the notification agent (Notification Web page or e-mail agent) when the performer completes the response to the notification. The procedure marks the notification as `'CLOSED'` and communicates RESPOND attributes back to the database via the callback function (if supplied).

This procedure also accepts the name of the individual who actually responded to the notification. This may be useful to know especially if the notification is assigned to a multi-user role. The information is stored in the `RESPONDER` column of the WF_NOTIFICATIONS table. The value stored in this column depends on how the user responds to the notification. The following table shows the value that is stored for each response mechanism.

Responder Values

Response Mechanism	Value Stored
Web	Web login username
E-mail	E-mail username as displayed in the mail response

Additionally, the *Respond()* procedure calls *NtfSignRequirementsMet()* to determine whether the response meets any signature requirements imposed by the electronic signature policy of the notification. If the requirements have not been met, *Respond()* raises an error. See: #WF_SIG_POLICY Attribute, *Oracle Workflow Developer's Guide* and NtfSignRequirementsMet, page 4-26.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

comment
An optional comment on the response

responder
The user who responded to the notification.

Responder

PL/SQL Syntax

```
function RESPONDER
  (nid in number)
  return varchar2;
```

Java Syntax

```
public static String responder
  (WFContext wCtx,
   BigDecimal nid)
```

Description

This function returns the responder of a closed notification.

If the notification was closed using the Web notification interface the value returned will be a valid role defined in the view WF_ROLES. If the notification was closed using the e-mail interface then the value returned will be an e-mail address. See: Respond, page 4-24.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

NtfSignRequirementsMet

PL/SQL Syntax

```
function NtfSignRequirementsMet
  (nid in number)
  return boolean;
```

Description

Returns `'TRUE'` if the response to a notification meets the signature requirements imposed by the electronic signature policy for the notification. See: #WF_SIG_POLICY Attribute, *Oracle Workflow Developer's Guide*.

- If the notification uses a signature policy that requires an electronic signature to validate a user's response, then a valid signature by a user who has authority to sign the response must be submitted in order for the response to meet the requirements. The signature must be of the appropriate type, either password-based or certificate-based, depending on the signature policy.
- If the notification uses the default policy, which does not require a signature, or if no signature policy is defined for the notification, then a response without a signature meets the requirements.

However, if the signature policy for the notification requires an electronic signature, but a valid signature has not been submitted, then the response does not meet the requirements. In this case *NtfSignRequirementsMet()* returns `'FALSE'`.

Arguments (input)

nid
The notification ID.

Related Topics

Respond, page 4-24

VoteCount

PL/SQL Syntax

```
procedure VoteCount
  (gid in number,
   ResultCode in varchar2,
   ResultCount out number,
   PercentOfTotalPop out number,
   PercentOfVotes out number);
```

Java Syntax

```
public static WFTwoDArray voteCount
  (WFContext wCtx,
   BigDecimal gid,
   String resultCode)
```

Description

Counts the number of responses for a specified result code.

Use this procedure only if you are writing your own custom Voting activity. See: Voting Activity, *Oracle Workflow Developer's Guide*.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

gid
The notification group ID.

ResultCode
Result code to be tallied.

OpenNotificationsExist

PL/SQL Syntax

```
function OpenNotificationsExist
  (gid in number)
  return boolean;
```

Java Syntax

```
public static boolean openNotificationsExist
  (WFContext wCtx,
   BigDecimal gid)
```

Description

This function returns 'TRUE' if any notification associated with the specified notification group ID is 'OPEN', otherwise it returns 'FALSE'.

Use this procedure only if you are writing your own custom Voting activity. See: Voting Activity, *Oracle Workflow Developer's Guide*.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

gid
The notification group ID.

Close

PL/SQL Syntax

```
procedure Close
  (nid in number,
   responder in varchar2 default null);
```

Java Syntax

```
public static boolean close
  (WFContext wCtx,
   BigDecimal nid,
   String responder)
```

Description

This procedure closes a notification.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

responder
The user or role who responded to the notification.

AddAttr

PL/SQL Syntax

```
procedure AddAttr
  (nid in number,
   aname in varchar2);
```

Java Syntax

```
public static boolean addAttr
  (WFContext wCtx,
   BigDecimal nid,
   String aName)
```

Description

Adds a new runtime notification attribute. You should perform validation and insure consistency in the use of the attribute, as it is completely unvalidated by Oracle Workflow.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

aname
The attribute name.

avalue
The attribute value.

Example

Example
The following code excerpt shows an example of how to call *addAttr()* in a Java program. The example code is from the `WFTest.java` program.

```
if (WFNotificationAPI.addAttr(ctx, myNid, myAttr) == false)
{
 System.out.println("Add attribute " + myAttr + " failed.");
}
```

SetAttribute

PL/SQL Syntax

```
procedure SetAttrText
  (nid in number,
   aname in varchar2,
   avalue in varchar2);

procedure SetAttrNumber
  (nid in number,
   aname in varchar2,
   avalue in number);

procedure SetAttrDate
  (nid in number,
   aname in varchar2,
   avalue in date);
```

Java Syntax

```
public static boolean setAttrText
  (WFContext wCtx,
   BigDecimal nid,
   String aName,
   String aValue)

public static boolean setAttrNumber
  (WFContext wCtx,
   BigDecimal nid,
   String aName,
   BigDecimal aValue)

public static boolean setAttrDate
  (WFContext wCtx,
   BigDecimal nid,
   String aName,
   String aValue)
```

Description

Used at both send and respond time to set the value of notification attributes. The notification agent (sender) may set the value of SEND attributes. The performer (responder) may set the value of RESPOND attributes.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

aname
The attribute name.

avalue
The attribute value.

Example

Example

The following code excerpt shows an example of how to call a *setAttribute* method in a Java program. The example code is from the `WFTest.java` program.

```
if (WFNotificationAPI.setAttrDate(ctx, myNid, myAttr, value)
    == false)
{
  System.out.println("set attribute " + myAttr + " to " +
    value + " failed.");
}
```

GetAttrInfo

PL/SQL Syntax

```
procedure GetAttrInfo
  (nid in number,
   aname in varchar2,
   atype out varchar2,
   subtype out varchar2,
   format out varchar2);
```

Java Syntax

```
public static WFTwoDArray getAttrInfo
  (WFContext wCtx,
   BigDecimal nid,
   String aName)
```

Description

Returns information about a notification attribute, such as its type, subtype, and format, if any is specified. The subtype is always `SEND` or `RESPOND` to indicate the attribute's source.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

aname
The attribute name.

Example

Example
The following code excerpt shows an example of how to call *getAttrInfo()* in a Java program. The example code is from the `WFTest.java` program.

```
dataSource = WFNotificationAPI.getAttrInfo(ctx, myNid,
    myAttr);
displayDataSource(ctx, dataSource);

// the first element is the attribute type
myAttrType = (String) dataSource.getData(0,0);
```

GetInfo

PL/SQL Syntax

```
procedure GetInfo
  (nid in number,
   role out varchar2,
   message_type out varchar2,
   message_name out varchar2,
   priority out number,
   due_date out date,
   status out varchar2);
```

Java Syntax

```
public static WFTwoDArray getInfo
  (WFContext wCtx,
   BigDecimal nid)
```

Description

Returns the role that the notification is sent to, the item type of the message, the name of the message, the notification priority, the due date and the status for the specified notification.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

Example

Example

The following code excerpt shows an example of how to call *getInfo()* in a Java program. The example code is from the `WFTest.java` program.

```
// Notification Info
System.out.println("Notification Info for nid " + myNid);
dataSource = WFNotificationAPI.getInfo(ctx, myNid);
displayDataSource(ctx, dataSource);
```

GetText

PL/SQL Syntax

```
function GetText
  (some_text in varchar2,
   nid in number,
   disptype in varchar2 default '')
  return varchar2;
```

Java Syntax

```
public static String getText
  (WFContext wCtx,
   String someText,
   BigDecimal nid,
   String dispType)
```

Description

Substitutes tokens in an arbitrary text string using token values from a particular notification. This function may return up to 32K characters. You cannot use this function in a view definition or in an Oracle Forms Developer form. For views and forms, use *GetShortText()* which truncates values at 1950 characters.

If an error is detected, this function returns `some_text` unsubstituted rather than raise exceptions.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

some_text or someText
Text to be substituted.

nid
Notification ID of notification to use for token values.

disptype or dispType
The display type of the message body that you are token substituting the text into. Valid display types are:

- `wf_notification.doc_text`, which returns `text/plain`
- `wf_notification.doc_html`, which returns `text/html`
- `wf_notification.doc_attach`, which returns `null`

The default is null.

GetShortText

PL/SQL Syntax

```
function GetShortText
  (some_text in varchar2,
   nid in number)
  return varchar2;
```

Description

Substitutes tokens in an arbitrary text string using token values from a particular notification. This function may return up to 1950 characters. This function is meant for use in view definitions and Oracle Forms Developer forms, where the field size is limited to 1950 characters. Use *GetText()* in other situations where you need to retrieve up to 32K characters.

If an error is detected, this function returns `some_text` unsubstituted rather than raise exceptions.

Arguments (input)

some_text
Text to be substituted.

nid
Notification ID of notification to use for token values.

GetAttribute

PL/SQL Syntax

```
function GetAttrText
  (nid in number,
   aname in varchar2)
  return varchar2;

function GetAttrNumber
  (nid in number,
   aname in varchar2)
  return number;

function GetAttrDate
  (nid in number,
   aname in varchar2)
  return date;
```

Java Syntax

```
public static String getAttrText
  (WFContext wCtx,
   BigDecimal nid,
   String aName)

public static BigDecimal getAttrNumber
  (WFContext wCtx,
   BigDecimal nid,
   String aName)

public static String getAttrDate
  (WFContext wCtx,
   BigDecimal nid,
   String aName)
```

Description

Returns the value of the specified message attribute.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

aname
The message attribute name.

Example

Example
The following code excerpt shows an example of how to call the *getAttribute* methods in a Java program. The example code is from the `WFTest.java` program.

```
// we get the value according to the type.
if (myAttrType == "DATE")
{
  value = WFNotificationAPI.getAttrDate(ctx, myNid, myAttr);
}
else if (myAttrType == "NUMBER")
{
  value = (WFNotificationAPI.getAttrNumber(ctx, myNid,
    myAttr)).toString();
}
else if (myAttrType == "DOCUMENT")
{
  value = WFNotificationAPI.getAttrDoc(ctx, myNid, myAttr,
    null);
}
else
  value = WFNotificationAPI.getAttrText(ctx, myNid, myAttr);

System.out.println(myAttr.toString() + " = '" + value +
    "'");
```

GetAttrDoc

PL/SQL Syntax

```
function GetAttrDoc
  (nid in number,
   aname in varchar2,
   disptype in varchar2)
  return varchar2;
```

Java Syntax

```
public static String getAttrDoc
  (WFContext wCtx,
   BigDecimal nid,
   String aName,
   String dispType)
```

Description

Returns the displayed value of a Document-type attribute. The referenced document appears in either plain text or HTML format, as requested.

If you wish to retrieve the actual attribute value, that is, the document key string instead of the actual document, use *GetAttrText()*.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

aname
The message attribute name.

disptype
The display type of the document you wish to return. Valid display types are:

- `wf_notification.doc_text`, which returns `text/plain`
- `wf_notification.doc_html`, which returns `text/html`
- `wf_notification.doc_attach`, which returns null

GetSubject

PL/SQL Syntax

```
function GetSubject
  (nid in number)
  return varchar2
```

Java Syntax

```
public static String getSubject
  (WFContext wCtx,
   BigDecimal nid)
```

Description

Returns the subject line for the notification message. Any message attribute in the subject is token substituted with the value of the corresponding message attribute.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

GetBody

PL/SQL Syntax

```
function GetBody
  (nid in number,
   disptype in varchar2 default '')
  return varchar2;
```

Java Syntax

```
public static String getBody
  (WFContext wCtx,
   BigDecimal nid,
   String dispType)
```

Description

Returns the HTML or plain text message body for the notification, depending on the message body type specified. Any message attribute in the body is token substituted with the value of the corresponding notification attribute. This function may return up to 32K characters. You cannot use this function in a view definition or in an Oracle Applications form. For views and forms, use *GetShortBody()* which truncates values at 1950 characters.

Note that the returned plain text message body is *not* formatted; it should be wordwrapped as appropriate for the output device. Body text may contain tabs (which indicate indentation) and newlines (which indicate paragraph termination).

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

disptype
The display type of the message body you wish to fetch. Valid display types are:

- `wf_notification.doc_text`, which returns `text/plain`
- `wf_notification.doc_html`, which returns `text/html`
- `wf_notification.doc_attach`, which returns null

The default is null.

GetShortBody

PL/SQL Syntax

```
function GetShortBody
  (nid in number)
  return varchar2;
```

Description

Returns the message body for the notification. Any message attribute in the body is token substituted with the value of the corresponding notification attribute. This function may return up to 1950 characters. This function is meant for use in view definitions and Oracle Forms Developer forms, where the field size is limited to 1950 characters. Use *GetBody()* in other situations where you need to retrieve up to 32K characters.

Note that the returned plain text message body is *not* formatted; it should be wordwrapped as appropriate for the output device. Body text may contain tabs (which indicate indentation) and newlines (which indicate paragraph termination).

If an error is detected, this function returns the body unsubstituted or null if all else fails, rather than raise exceptions.

Note: This function is intended for displaying messages in forms or views only.

Arguments (input)

nid
The notification ID.

TestContext

PL/SQL Syntax

```
function TestContext
  (nid in number)
  return boolean;
```

Description

Tests if the current context is correct by calling the Item Type Selector/Callback function. This function returns `TRUE` if the context check is OK, or if no Selector/Callback function is implemented. It returns `FALSE` if the context check fails.

Arguments (input)

nid
The notification ID.

AccessCheck

PL/SQL Syntax

```
function AccessCheck
  (access_str in varchar2)
  return varchar2;
```

Java Syntax

```
public static String accessCheck
  (WFContext wCtx,
   String accessString)
```

Description

Returns a username if the notification access string is valid and the notification is open, otherwise it returns null. The access string is automatically generated by the notification mailer that sends the notification and is used to verify the authenticity of both text and HTML versions of e-mail notifications.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

access_str or
accessString
The access string, in the format *nid/nkey* where *nid* is the notification ID and *nkey* is the notification key.

WorkCount

PL/SQL Syntax

```
function WorkCount
  (username in varchar2)
  return number;
```

Java Syntax

```
public static BigDecimal workCount
  (WFContext wCtx,
   String userName)
```

Description

Returns the number of open notifications assigned to a role.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

username
The internal name of a role.

getNotifications

Java Syntax

```
public static WFTwoDArray getNotifications
  (WFContext wCtx,
   String itemType,
   String itemKey)
```

Description

Returns a list of notifications for the specified item type and item key.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

itemType
The internal name of the item type.

itemKey
A string derived from the application object's primary key. The string uniquely identifies the item within the item type. The item type and key together identify the process instance.

getNotificationAttributes

Java Syntax

```
public static WFTwoDArray getNotificationAttributes
  (WFContext wCtx,
   BigDecimal nid)
```

Description

Returns a list of notification attributes and their corresponding values for the specified notification ID.

Arguments (input)

wCtx
Workflow context information. Required for the Java method only. See: Oracle Workflow Context, page 2-4.

nid
The notification ID.

Example

Example
The following code excerpt shows an example of how to call *getNotificationAttributes()* in a Java program. The example code is from the `WFTest.java` program.

```
// List available Notification Attributes
System.out.println("List of Attributes for id " + myNid +
    ":");
dataSource =
    WFNotificationAPI.getNotificationAttributes(ctx, myNid);
displayDataSource(ctx, dataSource);
```

WriteToClob

PL/SQL Syntax

```
procedure WriteToClob
  (clob_loc in out clob,
   msg_string in varchar2);
```

Description

Appends a character string to the end of a character large object (CLOB). You can use this procedure to help build the CLOB for a PL/SQL CLOB document attribute for a notification.

Arguments (input)

clob_loc
The CLOB to which the string should be added.

msg_string
A string of character data.

Related Topics

To Define a Document Attribute, *Oracle Workflow Developer's Guide*

"PL/SQL CLOB" Documents, *Oracle Workflow Developer's Guide*

Denormalize_Notification

PL/SQL Syntax

```
procedure Denormalize_Notification
  (nid in number,
   username in varchar2 default null,
   langcode in varchar2 default null);
```

Description

Stores denormalized values for certain notification fields, including the notification subject, in the WF_NOTIFICATIONS table. If you are using the Notification System to send a notification outside of a workflow process, you must call *Denormalize_Notification()* after setting the values for any notification attributes, in order to populate the denormalized fields.

Denormalize_Notification() tests whether the language in which the notification should be delivered matches the current session language, and stores the denormalized information according to this setting only if the languages match. You can indicate the language for the notification in a number of ways.

- If you specify a role name when you call the API, the language setting for that role is used to determine the notification language.
- If you do not specify a role name, you can specify a language code for the language you want.

  Note: If you specify both a role name and a language code, the role name is used to determine the notification language, and the language code is ignored.

- If you specify neither a role name nor a language code, the notification language defaults to the language setting for the recipient role of the notification.

If the notification language and the current session language do not match, the procedure does not store any denormalized information. In this case, the viewing interface through which the notification recipients access notifications must check the language and perform the denormalization. The Oracle Workflow Worklist will perform these tasks for you if your users access their notifications through the Worklist Web pages.

Arguments (input)

nid
The notification ID.

username
An optional internal name of a role used to determine the notification language.

langcode
An optional language code used to determine the notification language if no role name is provided.

SubstituteSpecialChars

PL/SQL Syntax

```
function SubstituteSpecialChars
  (some_text in varchar2)
  return varchar2;
```

Pragmas

```
pragma RESTRICT_REFERENCES(SubstituteSpecialChars, WNDS);
```

Description

Substitutes HTML character entity references for special characters in a text string and returns the modified text including the substitutions.

You can use this function as a security precaution when creating a PL/SQL document or a PL/SQL CLOB document that contains HTML, to ensure that only the HTML code you intend to include is executed. If you retrieve any data from the database at runtime for inclusion in the document, use *SubstituteSpecialChars()* to replace any HTML tag characters in that data, so that those characters will not be interpreted as HTML code and executed.

Note that you should not substitute entity references for HTML tags that you include in the document yourself. Otherwise, the document will not be displayed with your intended HTML formatting. You only need to perform this substitution for data that is retrieved from the database at runtime, which may be entered from an external source.

The following table shows each special character and the entity reference with which it is replaced.

Entity Reference Replacements for Special Characters

Character	Entity Reference
<	&lt;
>	&gt;
\	&#92;
&	&amp;
"	&quot;
'	&#39;

Arguments (input)

some_text
The text string in which you want to replace special characters.

Notification Mailer Utility API

The notification mailer utility API can be used to encode data in a binary large object (BLOB) to base64. This API is defined in a PL/SQL package called WF_MAIL_UTIL.

> **Note:** This package is only available if your database version is Oracle9*i* Database or higher. The Oracle8*i* Database does not support base64 encoding.

EncodeBLOB

PL/SQL Syntax

```
procedure EncodeBLOB
  (pIDoc in blob,
   pODoc in out nocopy clob);
```

Description

Encodes the specified BLOB to base64 and returns the encoded data as a character large object (CLOB). You can use this procedure to store a BLOB in a PL/SQL CLOB document to be included in a notification message.

> **Note:** This API is only available if your database version is Oracle9*i* Database or higher. The Oracle8*i* Database does not support base64 encoding.

Arguments (input)

pIDoc
The BLOB to encode.

pODoc
The CLOB in which the encoded data should be stored.

Related Topics

Standard APIs for "PL/SQL" Documents, *Oracle Workflow Developer's Guide*

5

Business Event System APIs

This chapter describes the APIs for the Oracle Workflow Business Event System. The APIs consist of datatypes and PL/SQL functions and procedures that you can use to access the Business Event System.

This chapter covers the following topics:

- Overview of the Oracle Workflow Business Event System
- Business Event System Datatypes
- Event APIs
- Event Subscription Rule Function APIs
- Event Function APIs
- Business Event System Replication APIs
- Business Event System Cleanup API

Overview of the Oracle Workflow Business Event System

The Oracle Workflow Business Event System leverages the Oracle Advanced Queuing infrastructure to communicate business events between systems. When a significant business event occurs in an internet or intranet application on a system, it triggers event subscriptions that specify the processing to execute for that event.

Subscriptions can include the following types of processing:

- Sending event information to a workflow process
- Sending event information to named communication points called agents on the local system or external systems
- Sending a notification to a role
- Receiving an Oracle XML Gateway message from a trading partner (Oracle E-Business Suite only)
- Sending an Oracle XML Gateway message to a trading partner (Oracle E-Business Suite only)
- Executing custom code on the event information

The event information communicated by the Business Event System is called an event message. The event message includes header properties to identify the event as well as event data describing what occurred.

You define events, systems, agents, and subscriptions in the Event Manager. You can also define event activities in the Workflow Builder to include business events in your workflow processes.

Related Topics

Business Event System Datatypes, page 5-2

Event APIs, page 5-20

Event Subscription Rule Function APIs, page 5-33

Event Function APIs, page 5-44

Business Event System Replication APIs, page 5-50

Business Event System Cleanup APIs, page 5-58

Managing Business Events, *Oracle Workflow Developer's Guide*

Event Activities, *Oracle Workflow Developer's Guide*

Business Event System Datatypes

Oracle Workflow uses a number of abstract datatypes (ADTs) to model the structure and behavior of Business Event System data. These datatypes include the following:

- Agent structure: WF_AGENT_T, page 5-2
- Parameter structure: WF_PARAMETER_T, page 5-4
- Parameter list structure: WF_PARAMETER_LIST_T, page 5-5
- Event message structure: WF_EVENT_T, page 5-6

The Business Event System datatypes are created by a script called `wftypes.sql`, which is located in the *`ORACLE_HOME`*`/wf/sql` directory for the standalone version of Oracle Workflow, or in the `$FND_TOP/sql` directory for the version of Oracle Workflow embedded in Oracle Applications.

See: User-Defined Datatypes, *Oracle Concepts*.

Related Topics

Example for Using Abstract Datatypes, page 5-17

Agent Structure

Oracle Workflow uses the object type `WF_AGENT_T` to store information about an agent in a form that can be referenced by an event message. The following table lists the attributes of the `WF_AGENT_T` datatype.

WF_AGENT_T Attributes

Attribute Name	Datatype	Description
NAME	VARCHAR2(30)	The name of the agent.
SYSTEM	VARCHAR2(30)	The system where the agent is located.

The WF_AGENT_T object type also includes the following methods, which you can use to retrieve and set the values of its attributes.

- getName, page 5-3
- getSystem, page 5-3
- setName, page 5-3
- setSystem, page 5-4

Related Topics

Agents, *Oracle Workflow Developer's Guide*

getName

PL/SQL Syntax

```
MEMBER FUNCTION getName
  return varchar2
```

Description

Returns the value of the NAME attribute in a WF_AGENT_T object.

getSystem

PL/SQL Syntax

```
MEMBER FUNCTION getSystem
  return varchar2
```

Description

Returns the value of the SYSTEM attribute in a WF_AGENT_T object.

setName

PL/SQL Syntax

```
MEMBER PROCEDURE setName
  (pName in varchar2)
```

Description

Sets the value of the NAME attribute in a WF_AGENT_T object.

Arguments (input)

pName
The value for the NAME attribute.

setSystem

PL/SQL Syntax

```
MEMBER PROCEDURE setSystem
    (pSystem in varchar2)
```

Description

Sets the value of the SYSTEM attribute in a WF_AGENT_T object.

Arguments (input)

pSystem
The value for the SYSTEM attribute.

Parameter Structure

Oracle Workflow uses the object type WF_PARAMETER_T to store a parameter name and value pair in a form that can be included in an event message parameter list. WF_PARAMETER_T allows custom values to be added to the WF_EVENT_T event message object. The following table lists the attributes of the WF_PARAMETER_T datatype.

WF_PARAMETER_T Attributes

Attribute Name	Datatype	Description
NAME	VARCHAR2(30)	The parameter name.
VALUE	VARCHAR2(2000)	The parameter value.

The WF_PARAMETER_T object type also includes the following methods, which you can use to retrieve and set the values of its attributes.

- getName, page 5-4
- getValue, page 5-5
- setName, page 5-5
- setValue, page 5-5

getName

PL/SQL Syntax

```
MEMBER FUNCTION getName
  return varchar2
```

Description

Returns the value of the NAME attribute in a WF_PARAMETER_T object.

getValue

PL/SQL Syntax

```
MEMBER FUNCTION getValue
  return varchar2
```

Description

Returns the value of the VALUE attribute in a WF_PARAMETER_T object.

setName

PL/SQL Syntax

```
MEMBER PROCEDURE setName
  (pName in varchar2)
```

Description

Sets the value of the NAME attribute in a WF_PARAMETER_T object.

Arguments (input)

pName
The value for the NAME attribute.

setValue

PL/SQL Syntax

```
MEMBER PROCEDURE setValue
   (pValue in varchar2)
```

Description

Sets the value of the VALUE attribute in a WF_PARAMETER_T object.

Arguments (input)

pValue
The value for the VALUE attribute.

Parameter List Structure

Oracle Workflow uses the named varying array (varray) WF_PARAMETER_LIST_T to store a list of parameters in a form that can be included in an event message. WF_PARAMETER_LIST_T allows custom values to be added to the WF_EVENT_T event message object. The WF_PARAMETER_LIST_T datatype can include up to 100 parameter name and value pairs. A description of this datatype is as follows:

WF_PARAMETER_LIST_T

- Maximum size: 100
- Element datatype: WF_PARAMETER_T

Event Message Structure

Oracle Workflow uses the object type WF_EVENT_T to store event messages. This datatype contains all the header properties of an event message as well as the event data payload, in a serialized form that is suitable for transmission outside the system.

WF_EVENT_T defines the event message structure that the Business Event System and the Workflow Engine use to represent a business event. Internally, the Business Event System and the Workflow Engine can only communicate events in this format. Many of the standard queues that Oracle Workflow provides for the Business Event System use WF_EVENT_T as their payload type.

> **Note:** If you want to use queues with a custom payload type, including any existing queues you already have defined on your system, you must create a queue handler to translate between the standard Workflow WF_EVENT_T structure and your custom payload type. See: Setting Up Queues, *Oracle Workflow Administrator's Guide* and Standard APIs for a Queue Handler, *Oracle Workflow Developer's Guide*.

The following table lists the attributes of the WF_EVENT_T datatype.

WF_EVENT_T Attributes

Attribute Name	Datatype	Description
PRIORITY	NUMBER	The priority with which the message recipient should dequeue the message. A smaller number indicates a higher priority. For example, 1 represents a high priority, 50 represents a normal priority, and 99 represents a low priority.
SEND_DATE	DATE	The date and time when the message is available for dequeuing. The send date can be set to the system date to indicate that the message is immediately available for dequeuing, or to a future date to indicate future availability. If the send date is set to a future date when an event is raised, the event message is placed on the WF_DEFERRED queue, and subscription processing does not begin until the specified date. If the send date is set to a future date when an event is sent to an agent, the event message is propagated to that agent's queue, but does not become available for the consumer to dequeue until the specified date.

Attribute Name	Datatype	Description
RECEIVE_DATE	DATE	The date and time when the message is dequeued by an agent listener.
CORRELATION_ID	VARCHAR2(240)	A correlation identifier that associates this message with other messages. This attribute is initially blank but can be set by a function. If a value is set for the correlation ID, then that value is used as the item key if the event is sent to a workflow process. Note that the item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.
PARAMETER_LIST	WF_PARAMETER_LIST_T	A list of additional parameter name and value pairs.
EVENT_NAME	VARCHAR2(240)	The internal name of the event.
EVENT_KEY	VARCHAR2(240)	The string that uniquely identifies the instance of the event.
EVENT_DATA	CLOB	A set of additional details describing what occurred in the event. The event data can be structured as an XML document.
FROM_AGENT	WF_AGENT_T	The agent from which the event is sent. For locally raised events, this attribute is initially null.
TO_AGENT	WF_AGENT_T	The agent to which the event should be sent (the message recipient).
ERROR_SUBSCRIPTION	RAW(16)	If an error occurs while processing this event, this is the subscription that was being executed when the error was encountered.
ERROR_MESSAGE	VARCHAR2(4000)	An error message that the Event Manager generates if an error occurs while processing this event.
ERROR_STACK	VARCHAR2(4000)	An error stack of arguments that the Event Manager generates if an error occurs while processing this event. The error stack provides context information to help you locate the source of an error.

The WF_EVENT_T object type also includes the following methods, which you can use to retrieve and set the values of its attributes.

- Initialize, page 5-9
- getPriority, page 5-9
- getSendDate, page 5-9
- getReceiveDate, page 5-10
- getCorrelationID, page 5-10
- getParameterList, page 5-10
- getEventName, page 5-10
- getEventKey, page 5-10
- getEventData, page 5-11
- getFromAgent, page 5-11
- getToAgent, page 5-11
- getErrorSubscription, page 5-11
- getErrorMessage, page 5-11
- getErrorStack, page 5-12
- setPriority, page 5-12
- setSendDate, page 5-12
- setReceiveDate, page 5-12
- setCorrelationID, page 5-13
- setParameterList, page 5-13
- setEventName, page 5-13
- setEventKey, page 5-13
- setEventData, page 5-14
- setFromAgent, page 5-14
- setToAgent, page 5-14
- setErrorSubscription, page 5-15
- setErrorMessage, page 5-15
- setErrorStack, page 5-15
- Content, page 5-15
- Address, page 5-16
- AddParameterToList, page 5-16
- GetValueForParameter, page 5-17

Note: You can set the values of the EVENT_NAME, EVENT_KEY, and EVENT_DATA attributes individually using the setEventName, setEventKey, and setEventData methods, or you can use the Content

method to set all three event content attributes at once. See: Content, page 5-15.

Similarly, you can set the values of the FROM_AGENT, TO_AGENT, PRIORITY, and SEND_DATE attributes individually using the setFromAgent, setToAgent, setPriority, and setSendDate methods, or you can use the Address method to set all four address attributes at once. See: Address, page 5-16.

Related Topics

Example for Using Abstract Datatypes, page 5-17

Mapping Between WF_EVENT_T and SYS.AQ$_JMS_TEXT_MESSAGE, page 5-18

Initialize

PL/SQL Syntax

```
STATIC PROCEDURE initialize
  (new_wf_event_t in out wf_event_t)
```

Description

Initializes a new WF_EVENT_T object by setting the PRIORITY attribute to 0, initializing the EVENT_DATA attribute to EMPTY using the *Empty_CLOB()* function, and setting all other attributes to NULL.

Important: You must call the Initialize method before you can perform any further manipulation on a new WF_EVENT_T object.

Arguments (input)

new_wf_event_t
The WF_EVENT_T object to initialize.

getPriority

PL/SQL Syntax

```
MEMBER FUNCTION getPriority
  return number
```

Description

Returns the value of the PRIORITY attribute in a WF_EVENT_T object.

getSendDate

PL/SQL Syntax

```
MEMBER FUNCTION getSendDate
  return date
```

Description

Returns the value of the SEND_DATE attribute in a WF_EVENT_T object.

getReceiveDate

PL/SQL Syntax

```
MEMBER FUNCTION getReceiveDate
  return date
```

Description

Returns the value of the RECEIVE_DATE attribute in a WF_EVENT_T object.

getCorrelationID

PL/SQL Syntax

```
MEMBER FUNCTION getCorrelationID
  return varchar2
```

Description

Returns the value of the CORRELATION_ID attribute in a WF_EVENT_T object.

getParameterList

PL/SQL Syntax

```
MEMBER FUNCTION getParameterList
  return wf_parameter_list_t
```

Description

Returns the value of the PARAMETER_LIST attribute in a WF_EVENT_T object.

getEventName

PL/SQL Syntax

```
MEMBER FUNCTION getEventName
  return varchar2
```

Description

Returns the value of the EVENT_NAME attribute in a WF_EVENT_T object.

getEventKey

PL/SQL Syntax

```
MEMBER FUNCTION getEventKey
  return varchar2
```

Description

Returns the value of the EVENT_KEY attribute in a WF_EVENT_T object.

getEventData

PL/SQL Syntax

```
MEMBER FUNCTION getEventData
  return clob
```

Description

Returns the value of the EVENT_DATA attribute in a WF_EVENT_T object.

getFromAgent

PL/SQL Syntax

```
MEMBER FUNCTION getFromAgent
  return wf_agent_t
```

Description

Returns the value of the FROM_AGENT attribute in a WF_EVENT_T object.

getToAgent

PL/SQL Syntax

```
MEMBER FUNCTION getToAgent
  return wf_agent_t
```

Description

Returns the value of the TO_AGENT attribute in a WF_EVENT_T object.

getErrorSubscription

PL/SQL Syntax

```
MEMBER FUNCTION getErrorSubscription
  return raw
```

Description

Returns the value of the ERROR_SUBSCRIPTION attribute in a WF_EVENT_T object.

getErrorMessage

PL/SQL Syntax

```
MEMBER FUNCTION getErrorMessage
  return varchar2
```

Description

Returns the value of the ERROR_MESSAGE attribute in a WF_EVENT_T object.

getErrorStack

PL/SQL Syntax

```
MEMBER FUNCTION getErrorStack
  return varchar2
```

Description

Returns the value of the ERROR_STACK attribute in a WF_EVENT_T object.

setPriority

PL/SQL Syntax

```
MEMBER PROCEDURE setPriority
  (pPriority in number)
```

Description

Sets the value of the PRIORITY attribute in a WF_EVENT_T object.

Arguments (input)

pPriority
The value for the PRIORITY attribute.

setSendDate

PL/SQL Syntax

```
MEMBER PROCEDURE setSendDate
  (pSendDate in date default sysdate)
```

Description

Sets the value of the SEND_DATE attribute in a WF_EVENT_T object.

Arguments (input)

pSendDate
The value for the SEND_DATE attribute.

setReceiveDate

PL/SQL Syntax

```
MEMBER PROCEDURE setReceiveDate
  (pReceiveDate in date default sysdate)
```

Description

Sets the value of the RECEIVE_DATE attribute in a WF_EVENT_T object.

Arguments (input)

pReceiveDate
The value for the RECEIVE_DATE attribute.

setCorrelationID

PL/SQL Syntax

```
MEMBER PROCEDURE setCorrelationID
    (pCorrelationID in varchar2)
```

Description

Sets the value of the CORRELATION_ID attribute in a WF_EVENT_T object.

Arguments (input)

pCorrelationID
The value for the CORRELATION_ID attribute.

setParameterList

PL/SQL Syntax

```
MEMBER PROCEDURE setParameterList
  (pParameterList in wf_parameter_list_t)
```

Description

Sets the value of the PARAMETER_LIST attribute in a WF_EVENT_T object.

Arguments (input)

pParameterList
The value for the PARAMETER_LIST attribute.

setEventName

PL/SQL Syntax

```
MEMBER PROCEDURE setEventName
  (pEventName in varchar2)
```

Description

Sets the value of the EVENT_NAME attribute in a WF_EVENT_T object.

Arguments (input)

pEventName
The value for the EVENT_NAME attribute.

setEventKey

PL/SQL Syntax

```
MEMBER PROCEDURE setEventKey
  (pEventKey in varchar2)
```

Description

Sets the value of the EVENT_KEY attribute in a WF_EVENT_T object.

Arguments (input)

pEventKey
The value for the EVENT_KEY attribute.

setEventData

PL/SQL Syntax

```
MEMBER PROCEDURE setEventData
   (pEventData in clob)
```

Description

Sets the value of the EVENT_DATA attribute in a WF_EVENT_T object.

Arguments (input)

pEventData
The value for the EVENT_DATA attribute.

setFromAgent

PL/SQL Syntax

```
MEMBER PROCEDURE setFromAgent
  (pFromAgent in wf_agent_t)
```

Description

Sets the value of the FROM_AGENT attribute in a WF_EVENT_T object.

Arguments (input)

pFromAgent
The value for the FROM_AGENT attribute.

setToAgent

PL/SQL Syntax

```
MEMBER PROCEDURE setToAgent
  (pToAgent in wf_agent_t)
```

Description

Sets the value of the TO_AGENT attribute in a WF_EVENT_T object.

Arguments (input)

pToAgent
The value for the TO_AGENT attribute.

setErrorSubscription

PL/SQL Syntax

```
MEMBER PROCEDURE setErrorSubscription
  (pErrorSubscription in raw)
```

Description

Sets the value of the ERROR_SUBSCRIPTION attribute in a WF_EVENT_T object.

Arguments (input)

pErrorSubscription
The value for the ERROR_SUBSCRIPTION attribute.

setErrorMessage

PL/SQL Syntax

```
MEMBER PROCEDURE setErrorMessage
  (pErrorMessage in varchar2)
```

Description

Sets the value of the ERROR_MESSAGE attribute in a WF_EVENT_T object.

Arguments (input)

pErrorMessage
The value for the ERROR_MESSAGE attribute.

setErrorStack

PL/SQL Syntax

```
MEMBER PROCEDURE setErrorStack
   (pErrorStack in varchar2)
```

Description

Sets the value of the ERROR_STACK attribute in a WF_EVENT_T object.

Arguments (input)

pErrorStack
The value for the ERROR_STACK attribute.

Content

PL/SQL Syntax

```
MEMBER PROCEDURE Content
  (pName in varchar2,
   pKey in varchar2,
   pData in clob)
```

Description

Sets the values of all the event content attributes in a WF_EVENT_T object, including EVENT_NAME, EVENT_KEY, and EVENT_DATA.

Arguments (input)

pName
The value for the EVENT_NAME attribute.

pKey
The value for the EVENT_KEY attribute.

pData
The value for the EVENT_DATA attribute.

Address

PL/SQL Syntax

```
MEMBER PROCEDURE Address
  (pOutAgent in wf_agent_t,
   pToAgent in wf_agent_t,
   pPriority in number,
   pSendDate in date)
```

Description

Sets the values of the all address attributes in a WF_EVENT_T object, including FROM_AGENT, TO_AGENT, PRIORITY, and SEND_DATE.

Arguments (input)

pOutAgent
The value for the FROM_AGENT attribute.

pToAgent
The value for the TO_AGENT attribute.

pPriority
The value for the PRIORITY attribute.

pSendDate
The value for the SEND_DATE attribute.

AddParameterToList

PL/SQL Syntax

```
MEMBER PROCEDURE AddParameterToList
  (pName in varchar2,
   pValue in varchar2)
```

Description

Adds a new parameter name and value pair to the list stored in the PARAMETER_LIST attribute of a WF_EVENT_T object. If a parameter with the specified name already exists in the parameter list, then the previous value of that parameter is overwritten with the specified value.

Arguments (input)

pName
The parameter name.

pValue
The parameter value.

GetValueForParameter

PL/SQL Syntax

```
MEMBER FUNCTION GetValueForParameter
  (pName in varchar2) return varchar2
```

Description

Returns the value of the specified parameter from the list stored in the PARAMETER_LIST attribute of a WF_EVENT_T object. This method begins at the end of the parameter list and searches backwards through the list. If no parameter with the specified name is found in the parameter list, then the method returns NULL.

Arguments (input)

pName
The parameter name.

Example for Using Abstract Datatypes

The following example shows some ways to use abstract datatype methods in a SQL script, including:

- Initializing a new event message structure with the Initialize method

  Important: You must call the Initialize method before you can perform any further manipulation on a new WF_EVENT_T object.

- Initializing a CLOB locator
- Writing a text variable into a CLOB variable
- Setting the content attributes of the event message structure with the Content method
- Setting the address attributes of the event message structure with the Address method

The example code is from the script wfevtenq.sql, which enqueues an event message on a queue using an override agent. See: Wfevtenq.sql, *Oracle Workflow Administrator's Guide*.

```
declare
l_overrideagent varchar2(30) := '&overrideagent';
l_overridesystem varchar2(30) := '&overridesystem';
l_fromagent varchar2(30) := '&fromagent';
l_fromsystem varchar2(30) := '&fromsystem';
l_toagent varchar2(30) := '&toagent';
l_tosystem varchar2(30) := '&tosystem';
l_eventname varchar2(100) := '&eventname';
l_eventkey varchar2(100) := '&eventkey';
l_msg varchar2(200) := '&message';
l_clob clob;
l_overrideagent_t wf_agent_t;
l_toagent_t wf_agent_t;
l_fromagent_t wf_agent_t;
l_event_t wf_event_t;

begin

  /*You must call wf_event_t.initialize before you can manipulate
    a new wf_event_t object.*/
  wf_event_t.initialize(l_event_t);
  l_overrideagent_t := wf_agent_t(l_overrideagent, l_overridesyste
m);
  l_toagent_t := wf_agent_t(l_toagent, l_tosystem);
  l_fromagent_t := wf_agent_t(l_fromagent, l_fromsystem);

  if l_msg is null then
    l_event_t.Content(l_eventname, l_eventkey, null);
  else
    dbms_lob.createtemporary(l_clob, FALSE, DBMS_LOB.CALL);
    dbms_lob.write(l_clob, length(l_msg), 1, l_msg);
    l_event_t.Content(l_eventname, l_eventkey, l_clob);
  end if;

  l_event_t.Address(l_fromagent_t, l_toagent_t, 50, sysdate);

  wf_event.enqueue(l_event_t, l_overrideagent_t);

end;
```

Mapping Between WF_EVENT_T and SYS.AQ$_JMS_TEXT_MESSAGE

Java Message Service (JMS) is a messaging standard defined by Sun Microsystems, Oracle, IBM, and other vendors. JMS is a set of interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprise messaging product.

Oracle Java Message Service provides a Java API for Oracle Advanced Queuing (AQ) based on the JMS standard. Oracle JMS supports the standard JMS interfaces and has extensions to support the AQ administrative operations and other AQ features that are not a part of the standard. The abstract datatype used to store a JMS Text message in an AQ queue is called `SYS.AQ$_JMS_TEXT_MESSAGE`.

Oracle Workflow supports communication of JMS Text messages through the Business Event System by providing a queue handler called `WF_EVENT_OJMSTEXT_QH`. This queue handler translates between the standard Workflow `WF_EVENT_T` message

structure and SYS.AQ$_JMS_TEXT_MESSAGE. Oracle Workflow also provides standard inbound and outbound queues that you can use for JMS Text messages, These queues are called WF_JMS_IN and WF_JMS_OUT, respectively, and use the WF_EVENT_OJMSTEXT_QH queue handler. See: Agents, *Oracle Workflow Developer's Guide*.

The SYS.AQ$_JMS_TEXT_MESSAGE datatype contains the following attributes.

- HEADER - Header properties in the SYS.AQ$_JMS_HEADER datatype
- TEXT_LEN - The size of the message payload, set automatically
- TEXT_VC - The message payload in VARCHAR2 format, if the payload is equal to or less than 4000 bytes
- TEXT_LOB - The message payload in CLOB format, if the payload is greater than 4000 bytes

The SYS.AQ$_JMS_HEADER datatype contains the following attributes.

- REPLYTO - A Destination supplied by a client when a message is sent
- TYPE - The type of the message
- USERID - The identity of the user sending the message
- APPID - The identity of the application sending the message
- GROUPID - The identity of the message group of which this message is a part; set by the client
- GROUPSEQ - The sequence number of the message within the group
- PROPERTIES - Additional message properties in the SYS.AQ$_JMS_USERPRO PARRAY datatype

The SYS.AQ$_JMS_USERPROPARRAY datatype is a named varying array with a maximum size of 100. The datatype of its elements is another ADT named SYS.AQ$_JMS_USERPROPERTY.

The following table shows how the attributes of the WF_EVENT_T message structure are mapped to the attributes within the SYS.AQ$_JMS_TEXT_MESSAGE structure.

Mapping WF_EVENT_T Attributes to SYS.AQ$_JMS_TEXT_MESSAGE Attributes

WF_EVENT_T	SYS.AQ$_JMS_TEXT_MESSAGE
WF_EVENT_T.PRIORITY	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.SEND_DATE	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.RECEIVE_DATE	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.CORRELATION_ID	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.EVENT_NAME	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.EVENT_KEY	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.EVENT_DATA	TEXT_VC or TEXT_LOB
WF_EVENT_T.PARAMETER_LIST	SYS.AQ$_JMS_HEADER.REPLYTO
WF_EVENT_T.PARAMETER_LIST	SYS.AQ$_JMS_HEADER.TYPE
WF_EVENT_T.PARAMETER_LIST	SYS.AQ$_JMS_HEADER.USERID
WF_EVENT_T.PARAMETER_LIST	SYS.AQ$_JMS_HEADER.APPID
WF_EVENT_T.PARAMETER_LIST	SYS.AQ$_JMS_HEADER.GROUPID
WF_EVENT_T.PARAMETER_LIST	SYS.AQ$_JMS_HEADER.GROUPSEQ
WF_EVENT_T.PARAMETER_LIST (any parameters other than JMS header properties)	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.FROM_AGENT	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.TO_AGENT	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.ERROR_SUBSCRIPTION	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.ERROR_MESSAGE	SYS.AQ$_JMS_USERPROPARRAY
WF_EVENT_T.ERROR_STACK	SYS.AQ$_JMS_USERPROPARRAY

See: Using Oracle Java Message Service to Access AQ, *Oracle Application Developer's Guide - Advanced Queuing* or Using Oracle Java Message Service (OJMS) to Access Oracle Streams AQ, *Oracle Streams Advanced Queuing User's Guide and Reference* and Package oracle.jms, *Oracle Supplied Java Packages Reference*.

Event APIs

The Event APIs can be called by an application program or a workflow process in the runtime phase to communicate with the Business Event System and manage events. These APIs are defined in a PL/SQL package called WF_EVENT.

Raise

PL/SQL Syntax

```
procedure Raise
  (p_event_name in varchar2,
   p_event_key in varchar2,
   p_event_data in clob default NULL,
   p_parameters in wf_parameter_list_t default NULL,
   p_send_date in date default NULL);
```

Description

Raises a local event to the Event Manager. *Raise()* creates a `WF_EVENT_T` structure for this event instance and sets the specified event name, event key, event data, parameter list, and send date into the structure.

The event data can be passed to the Event Manager within the call to the *Raise()* API, or the Event Manager can obtain the event data itself by calling the generate function for the event, after first checking whether the event data is required by a subscription. If the event data is not already available in your application, you can improve performance by allowing the Event Manager to run the generate function and generate the event data only when subscriptions exist that require that data, rather than always generating the event data from your application at runtime. See: Events, *Oracle Workflow Developer's Guide* and Standard API for an Event Data Generate Function, *Oracle Workflow Developer's Guide*.

The send date can optionally be set to indicate when the event should become available for subscription processing. If the send date is null, *Raise()* sets the send date to the current system date. You can defer an event by setting the send date to a date later than the system date. In this case, the Event Manager places the event message on the standard WF_DEFERRED queue, where it remains in a `WAIT` state until the send date. When the send date arrives, the event message becomes available for dequeuing and will be dequeued the next time an agent listener runs on the WF_DEFERRED queue.

Note: If an event is deferred when it is raised, the event retains its original Local source type when it is dequeued from the WF_DEFERRED queue.

When an event is raised and is not deferred, or when an event that was deferred is dequeued from the WF_DEFERRED queue, the Event Manager begins subscription processing for the event. The Event Manager searches for and executes any enabled subscriptions by the local system to that event with a source type of Local, and also any enabled subscriptions by the local system to the Any event with a source type of Local. If no enabled subscriptions exist for the event that was raised (apart from subscriptions to the Any event), then Oracle Workflow executes any enabled subscriptions by the local system to the Unexpected event with a source type of Local.

Note: The Event Manager does not raise an error if the event is not defined.

The Event Manager checks each subscription before executing it to determine whether the subscription requires the event data. If the event data is required but is not already provided, the Event Manager calls the generate function for the event to produce the event data. If the event data is required but no generate function is defined for the event, Oracle Workflow creates a default set of event data using the event name and event key.

Note: Any exceptions raised during *Raise()* processing are not trapped, but instead are exposed to the code that called the *Raise()* procedure. This behavior enables you to use subscriptions and their rule functions to perform validation, with the same results as if the validation logic were coded inline.

Arguments (input)

p_event_name
The internal name of the event.

p_event_key
A string generated when the event occurs within a program or application. The event key uniquely identifies a specific instance of the event.

p_event_data
An optional set of information about the event that describes what occurred. The Event Manager checks each subscription before executing it to determine whether the subscription requires the event data. If the event data is required but is not already provided, the Event Manager calls the generate function for the event to produce the event data. See: Events, *Oracle Workflow Developer's Guide* and Standard API for an Event Data Generate Function, *Oracle Workflow Developer's Guide*.

p_parameters
An optional list of additional parameter name and value pairs.

p_send_date
An optional date to indicate when the event should become available for subscription processing.

Example

Example

```
declare
  l_xmldocument  varchar2(32000);
  l_eventdata  clob;
  l_parameter_list wf_parameter_list_t;
  l_message  varchar2(10);

begin

/*
** If the complete event data is easily available, we can
** optionally test if any subscriptions to this event
** require it (rule data = Message).
*/

  l_message := wf_event.test('<EVENT_NAME>');

/*
** If we do require a message, and we have the message now,
** set it; else we can just rely on the Event Generate
** Function callback code. Then raise the event with the
** required parameters.
*/

  if l_message = 'MESSAGE' then
    if l_xmldocument is not null then
      dbms_lob.createtemporary(l_eventdata, FALSE, DBMS_LOB.CALL);
      dbms_lob.write(l_eventdata, length(l_xmldocument), 1 ,
        l_xmldocument);
      -- Raise the Event with the message
      wf_event.raise( p_event_name => '<EVENT_NAME>',
        p_event_key  => '<EVENT_KEY>',
        p_event_data => l_eventdata,
        p_parameters => l_parameter_list);
    else
      -- Raise the Event without the message
      wf_event.raise( p_event_name => '<EVENT_NAME>',
        p_event_key  => '<EVENT_KEY>',
        p_parameters => l_parameter_list);
    end if;
  elsif
    l_message = 'KEY' then
      -- Raise the Event
      wf_event.raise( p_event_name => <EVENT_NAME>,
        p_event_key  => <EVENT_KEY>,
        p_parameters => l_parameter_list);
  end if;

/*
** Up to your own custom code to commit the transaction
*/

  commit;

/*
** Up to your own custom code to handle any major exceptions
```

```
*/

exception
when others then
null;
end;
```

Related Topics

Any Event, *Oracle Workflow Developer's Guide*

Unexpected Event, *Oracle Workflow Developer's Guide*

Raise3

PL/SQL Syntax

```
procedure Raise3
  (p_event_name in varchar2,
   p_event_key in varchar2,
   p_event_data in clob default NULL,
   p_parameter_list in out nocopy wf_parameter_list_t,
   p_send_date in date default NULL);
```

Description

Raises a local event to the Event Manager and returns the parameter list for the event. *Raise3()* performs the same processing as the *Raise()* procedure, except that *Raise3()* passes the event parameter list back to the calling application after completing the event subsription processing. See: Raise, page 5-21.

Raise3() creates a `WF_EVENT_T` structure for this event instance and sets the specified event name, event key, event data, parameter list, and send date into the structure. Then, if the event is not deferred, the Event Manager begins subscription processing for the event. The Event Manager searches for and executes any enabled subscriptions by the local system to that event with a source type of Local, and also any enabled subscriptions by the local system to the Any event with a source type of Local. If no enabled subscriptions exist for the event that was raised (apart from subscriptions to the Any event), then Oracle Workflow executes any enabled subscriptions by the local system to the Unexpected event with a source type of Local.

After completing subscription processing for the event, *Raise3()* returns the parameter list for the event, including any modifications made to the parameters by the rule functions of the subscriptions. In this way, event subscriptions can communicate parameters back to the application that raised the event.

> **Note:** Any exceptions raised during *Raise3()* processing are not trapped, but instead are exposed to the code that called the *Raise3()* procedure. This behavior enables you to use subscriptions and their rule functions to perform validation, with the same results as if the validation logic were coded inline.

Arguments (input)

p_event_name
The internal name of the event.

p_event_key
A string generated when the event occurs within a program or application. The event key uniquely identifies a specific instance of the event.

p_event_data
An optional set of information about the event that describes what occurred. The Event Manager checks each subscription before executing it to determine whether the subscription requires the event data. If the event data is required but is not already provided, the Event Manager calls the generate function for the event to produce the event data. See: Events, *Oracle Workflow Developer's Guide* and Standard API for an Event Data Generate Function, *Oracle Workflow Developer's Guide*.

p_parameter_list
A list of additional parameter name and value pairs.

p_send_date
An optional date to indicate when the event should become available for subscription processing.

Send

PL/SQL Syntax

```
procedure Send
   (p_event in out wf_event_t);
```

Description

Sends an event message from one agent to another. If the event message contains both a From Agent and a To Agent, the message is placed on the outbound queue of the From Agent and then asynchronously delivered to the To Agent by AQ propagation, or whichever type of propagation is implemented for the agents' protocol.

If the event message contains a To Agent but no specified From Agent, the message is sent from the default outbound agent that matches the queue type of the To Agent.

If the event message contains a From Agent but no specified To Agent, the event message is placed on the From Agent's queue without a specified recipient.

- You can omit the To Agent if the From Agent uses a multi-consumer queue with a subscriber list. (The standard Workflow queue handlers work only with multi-consumer queues.) In this case, the queue's subscriber list determines which consumers can dequeue the message. If no subscriber list is defined for that queue, however, the event message is placed on the WF_ERROR queue for error handling.

  Note: The subscriber list for a multi-consumer queue in Oracle Advanced Queuing is different from event subscriptions in the Oracle Workflow Business Event System. For more information, see: Subscription and Recipient Lists, *Oracle Application Developer's Guide - Advanced Queuing* or *Oracle Streams Advanced Queuing User's Guide and Reference*.

- You can also omit the To Agent if the From Agent uses a single-consumer queue for which you have defined a custom queue handler. For a single-consumer queue, no specified consumer is required.

The send date within the event message indicates when the message should become available for the consumer to dequeue. If the send date is blank, the *Send()* procedure resets the value to the current system date, meaning the message is immediately available for dequeuing as soon as it is propagated. If the send date is a future date, the message is marked with a delay time corresponding to that date and does not become available for dequeuing until the delay time has passed. For more information, see: Time Specification: Delay, *Oracle Application Developer's Guide - Advanced Queuing* or *Oracle Streams Advanced Queuing User's Guide and Reference*.

> **Note:** If you want to use the send date to determine when a message becomes available for dequeuing on the To Agent, you should set the send date during subscription processing before *Send()* is called.

Send() returns the final event message that was sent, including any properties set by the procedure.

Arguments (input)

p_event
The event message.

NewAgent

PL/SQL Syntax

```
function NewAgent
  (p_agent_guid in raw) return wf_agent_t;
```

Description

Creates a `WF_AGENT_T` structure for the specified agent and sets the agent's system and name into the structure. See: Agent Structure, page 5-2.

Arguments (input)

p_agent_guid
The globally unique identifier of the agent.

Test

PL/SQL Syntax

```
function Test
  (p_event_name in varchar2) return varchar2;
```

Description

Tests whether the specified event is enabled and whether there are any enabled subscriptions by the local system referencing the event, or referencing an enabled event

group that contains the event. *Test()* returns the most costly data requirement among these subscriptions, using the following result codes:

- NONE - No enabled local subscriptions reference the event, or the event does not exist.
- KEY - At least one enabled local subscription references the event, but all such subscriptions require only the event key.
- MESSAGE - At least one enabled local subscription on the event requires the complete event data.

Arguments (input)

p_event_name
The internal name of the event.

Enqueue

PL/SQL Syntax

```
procedure Enqueue
  (p_event in wf_event_t,
   p_out_agent_override in wf_agent_t default null);
```

Description

Enqueues an event message onto a queue associated with an outbound agent. You can optionally specify an override agent where you want to enqueue the event message. Otherwise, the event message is enqueued on the From Agent specified within the message. The message recipient is set to the To Agent specified in the event message. *Enqueue()* uses the queue handler for the outbound agent to place the message on the queue.

Arguments (input)

p_event
The event message.

p_out_agent_override
The outbound agent on whose queue the event message should be enqueued.

Listen

PL/SQL Syntax

```
procedure Listen
  (p_agent_name in varchar2,
   p_wait in binary_integer default dbms_aq.no_wait,
   p_correlation in varchar2 default null,
   p_deq_condition in varchar2 default null);
```

Description

Monitors an agent for inbound event messages and dequeues messages using the agent's queue handler, in the database tier.

The standard WF_EVENT_QH queue handler sets the date and time when an event message is dequeued into the RECEIVE_DATE attribute of the event message. Custom queue handlers can also set the RECEIVE_DATE value if this functionality is included in the Dequeue API.

When an event is dequeued, the Event Manager searches for and executes any enabled subscriptions by the local system to that event with a source type of External, and also any enabled subscriptions by the local system to the Any event with a source type of External. If no enabled subscriptions exist for the event that was received (apart from subscriptions to the Any event), then Oracle Workflow executes any enabled subscriptions by the local system to the Unexpected event with a source type of External.

The *Listen()* procedure exits after all event messages on the agent's queue have been dequeued, unless you specify a wait period to block on the queue waiting for additional messages.

You must not call *Listen()* from within application code. If you want to call this procedure directly, you can run it from SQL*Plus. Otherwise, you can schedule PL/SQL agent listeners for your inbound agents from Oracle Applications Manager or Oracle Enterprise Manager, depending on your version of Oracle Workflow. See: Scheduling Listeners for Local Inbound Agents, *Oracle Workflow Administrator's Guide*.

You can optionally restrict the event messages that the *Listen()* procedure will process by specifying an AQ correlation ID consisting of an event name, or a partial event name followed by a percent sign (%) as a wildcard character. Additionally, if your database version is Oracle9*i* Database or higher, you can also optionally restrict the event messages that the *Listen()* procedure will process by specifying a dequeue condition that references the properties or content of the message. However, you cannot specify both of these parameters at the same time. If you specify one, you must leave the other null.

Arguments (input)

p_agent_name
The name of the inbound agent.

p_wait
An optional wait period, in seconds, during which you want the agent listener to block on the agent's queue to wait for messages. By default an agent listener does not wait but exits after all messages on the queue have been dequeued.

p_correlation
Optionally specify an AQ correlation ID to identify the event messages that you want the agent listener to process. The AQ correlation ID for an event message in the Business Event System is usually specified as an event name, or as a partial event name followed by a percent sign (%) as a wildcard character. Consequently, by specifying an AQ correlation ID in this parameter, you can dedicate the agent listener to listen only for messages that are instances of the specified event or events. For example, you can specify oracle.apps.wf.notification% to listen for all events related to notifications whose names begin with that value. The default value for this correlation ID is null, which allows the agent listener to process messages that are instances of any event.

If a dequeue condition is specified in the next parameter, this parameter must be null.

See: Dequeue Methods, *Oracle Application Developer's Guide - Advanced Queuing* or *Oracle Streams Advanced Queuing User's Guide and Reference*.

> **Note:** The AQ correlation ID is different than the correlation ID contained within the WF_EVENT_T event message structure.

p_deq_condition
Optionally specify a dequeue condition to identify the event messages that you want the agent listener to process. A dequeue condition is an expression that is similar in syntax to the `WHERE` clause of a SQL query. Dequeue conditions are expressed in terms of the attributes that represent message properties or message content. The messages in the queue are evaluated against the condition, so you can restrict the agent listener to listen only for messages that satisfy this condition. The default value is null, which does not place any restriction on the messages the agent listener can process.

If an AQ correlation ID is specified in the previous parameter, this parameter must be null.

See: Dequeue Methods, *Oracle Application Developer's Guide - Advanced Queuing* or *Oracle Streams Advanced Queuing User's Guide and Reference*.

Related Topics

Any Event, *Oracle Workflow Developer's Guide*

Unexpected Event, *Oracle Workflow Developer's Guide*

Wfagtlst.sql, *Oracle Workflow Administrator's Guide*

Standard APIs for a Queue Handler, *Oracle Workflow Developer's Guide*

SetErrorInfo

PL/SQL Syntax

```
procedure SetErrorInfo
  (p_event in out wf_event_t,
   p_type in varchar2);
```

Description

Retrieves error information from the error stack and sets it into the event message. The error message and error stack are set into the corresponding attributes of the event message. The error name and error type are added to the `PARAMETER_LIST` attribute of the event message.

Arguments (input)

p_event
The event message.

p_type
The error type, either `'ERROR'` or `'WARNING'`.

SetDispatchMode

PL/SQL Syntax

```
procedure SetDispatchMode
  (p_mode in varchar2);
```

Description

Sets the dispatch mode of the Event Manager to either deferred or synchronous subscription processing. Call *SetDispatchMode()* with the mode `'ASYNC'` just before calling *Raise()* to defer all subscription processing forever for the event that you will raise. In this case, the Event Manager places the event on the WF_DEFERRED queue before executing any subscriptions for that event. The subscriptions are not executed until an agent listener runs to dequeue the event from the WF_DEFERRED queue.

You can call *SetDispatchMode()* with the mode `'SYNC'` to set the dispatch mode back to normal synchronous subscription processing. In this mode, the phase number for each subscription determines whether the subscription is executed immediately or deferred.

> **Note:** This method of deferring subscription processing is not recommended and should only be used in exceptional circumstances, since it requires hard-coding the deferral in your application. To retain the flexibility to modify subscription processing without intrusion into the application, you can simply mark some or all of the individual subscriptions for deferral using the subscription phase numbers.

Arguments (input)

p_mode
The dispatch mode: either `'ASYNC'` for deferred (asynchronous) subscription processing, or `'SYNC'` for synchronous subscription processing.

Related Topics

Deferred Subscription Processing, *Oracle Workflow Developer's Guide*

Raise, page 5-21

AddParameterToList

PL/SQL Syntax

```
procedure AddParameterToList
  (p_name in varchar2,
   p_value in varchar2,
   p_parameterlist in out wf_parameter_list_t);
```

Description

Adds the specified parameter name and value pair to the end of the specified parameter list varray. If the varray is null, *AddParameterToList()* initializes it with the new parameter.

Arguments (input)

p_name
The parameter name.

p_value
The parameter value.

p_parameterlist
The parameter list.

AddParameterToListPos

PL/SQL Syntax

```
procedure AddParameterToListPos
  (p_name in varchar2,
   p_value in varchar2,
   p_position out integer,
   p_parameterlist in out wf_parameter_list_t);
```

Description

Adds the specified parameter name and value pair to the end of the specified parameter list varray. If the varray is null, *AddParameterToListPos()* initializes it with the new parameter. The procedure also returns the index for the position at which the parameter is stored within the varray.

Arguments (input)

p_name
The parameter name.

p_value
The parameter value.

p_parameterlist
The parameter list.

GetValueForParameter

PL/SQL Syntax

```
function GetValueForParameter
  (p_name in varchar2,
   p_parameterlist in wf_parameter_list_t)
  return varchar2;
```

Description

Retrieves the value of the specified parameter from the specified parameter list varray. *GetValueForParameter()* begins at the end of the parameter list and searches backwards through the list.

Arguments (input)

p_name
The parameter name.

p_parameterlist
The parameter list.

GetValueForParameterPos

PL/SQL Syntax

```
function GetValueForParameterPos
  (p_position in integer,
   p_parameterlist in wf_parameter_list_t)
  return varchar2;
```

Description

Retrieves the value of the parameter stored at the specified position in the specified parameter list varray.

Arguments (input)

p_position
The index representing the position of the parameter within the parameter list.

p_parameterlist
The parameter list.

SetMaxNestedRaise

PL/SQL Syntax

```
procedure SetMaxNestedRaise
  (maxcount in number default 100);
```

Description

Sets the maximum number of nested raises that can be performed to the specified value. A nested raise occurs when one event is raised and a Local subscription to that event is executed and raises another event. The default maximum is 100.

Arguments (input)

max_count
The maximum number of nested raises to allow.

GetMaxNestedRaise

PL/SQL Syntax

```
function GetMaxNestedRaise
  return number;
```

Description

Returns the maximum number of nested raises that can currently be performed. A nested raise occurs when one event is raised and a Local subscription to that event is executed and raises another event.

Event Subscription Rule Function APIs

The event subscription rule function APIs provide standard rule functions that you can assign to event subscriptions. A rule function specifies the processing that Oracle Workflow performs when the subscription's triggering event occurs.

Oracle Workflow provides a standard Default_Rule function to perform basic subscription processing. The default rule function includes the following actions:

- Sending the event message to a workflow process, if specified in the subscription definition
- Sending the event message to an agent, if specified in the subscription definition

Oracle Workflow also provides some other standard rule functions that you can use. The Log, Error, Warning, and Success functions can be used for testing and debugging your application. Other standard rule functions provide specialized processing used in predefined Oracle Workflow event subscriptions or in special options you can choose to refine your subscription processing.

These rule function APIs are defined in a PL/SQL package called WF_RULE.

Related Topics

Event Subscriptions, *Oracle Workflow Developer's Guide*

Standard API for an Event Subscription Rule Function, *Oracle Workflow Developer's Guide*

Default_Rule

PL/SQL Syntax

```
function Default_Rule
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Performs default subscription processing for an event subscription. The default processing includes:

- Sending the event message to a workflow process, if specified in the subscription definition
- Sending the event message to an agent, if specified in the subscription definition

If either of these operations raises an exception, *Default_Rule()* traps the exception, stores the error information in the event message, and returns the status code `ERROR`. Otherwise, *Default_Rule()* returns the status code `SUCCESS`.

> **Note:** If the event message is being sent to the Default Event Error workflow process by the subscription, *Default_Rule()* generates a new correlation ID to use as the item key for the process in order to ensure that the item key is unique.

If you want to run a custom rule function on the event message before it is sent, you can define one subscription with a low phase number that uses the custom rule function, and then define another subscription with a higher phase number that uses the default rule function to send the event.

For example, follow these steps:

1. Define a subscription to the relevant event with your custom rule function and a phase of 10.
2. Define another subscription to the event with the rule function *WF_EVENT.Default_Rule* and a phase of 20, and specify the workflow or agent to which you want to send the event.
3. Raise the event to trigger the subscriptions. The subscription with the lower phase number will be executed first and will run your custom rule function on the event message. When the event is passed to the second subscription, the modified event message will be sent to the workflow or agent you specified.

You can also call *Default_Rule()* to add the default send processing within a custom rule function. If you enter a rule function other than *Default_Rule()* for a subscription, Oracle Workflow does not automatically send the event message to the workflow and agent specified in the subscription. Instead, if you want to send the message from the same subscription, you must explicitly include the send processing in your custom rule function, which you can optionally accomplish by calling *Default_Rule()*. See: Standard API for an Event Subscription Rule Function, *Oracle Workflow Developer's Guide*.

> **Note:** You may find it advantageous to define multiple subscriptions to an event with simple rule functions that you can reuse, rather than creating complex specialized rule functions that cannot be reused.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Log

PL/SQL Syntax

```
function Log
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Logs the contents of the specified event message using *DBMS_OUTPUT.put_line* and returns the status code `SUCCESS`. Use this function to output the contents of an event message to a SQL*Plus session for testing and debugging purposes.

For example, if you want to test a custom rule function that modifies the event message, you can use *Log()* to show the event message both before and after your custom rule function is run. Define three subscriptions to the relevant event as follows:

- Define the first subscription with a phase of 10 and the rule function *WF_RULE.Log*.
- Define the second subscription with a phase of 20 and your custom rule function.
- Define the third subscription with a phase of 30 and the rule function *WF_RULE.Log*.

Next, connect to SQL*Plus. Execute the following command:

```
set serveroutput on size 100000
```

Then raise the event using *WF_EVENT.Raise*. As the Event Manager executes your subscriptions to the event in phase order, you should see the contents of the event message both before and after your custom rule function is run.

> **Note:** You should not assign *Log()* as the rule function for any enabled subscriptions in a production instance of Oracle Workflow. This function should be used for debugging only.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Error

PL/SQL Syntax

```
function Error
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Returns the status code `ERROR`. Additionally, when you assign this function as the rule function for a subscription, you must enter a text string representing the internal name of an error message in the Parameters field for the subscription. When the

subscription is executed, *Error()* will set that error message into the event message using *setErrorMessage()*. See: setErrorMessage, page 5-15.

The text string you enter in the Parameters field must be a valid name of an Oracle Workflow error message. The names of the error messages provided by Oracle Workflow are stored in the NAME column of the WF_RESOURCES table for messages with a type of WFERR.

You can use *Error()* as a subscription rule function if you want to send the system administrator an error notification with one of the predefined Workflow error messages whenever a particular event is raised.

For example, define a subscription to the relevant event with the rule function *WF_RULE.Error* and enter WFSQL_ARGS in the Parameters field. Then raise the event to trigger the subscription. Because *Error()* returns the status code ERROR, the Event Manager places the event message on the WF_ERROR queue and subscription processing for the event is halted. When the listener runs on the WF_ERROR queue, an error notification will be sent to the system administrator with the message "Invalid value(s) passed for arguments", which is the display name of the WFSQL_ARGS error message.

> **Note:** *Error()* does not raise any exception to the calling application when it completes normally.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Warning

PL/SQL Syntax

```
function Warning
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Returns the status code WARNING. Additionally, when you assign this function as the rule function for a subscription, you must enter a text string representing the internal name of an error message in the Parameters field for the subscription. When the subscription is executed, *Warning()* will set that error message into the event message using *setErrorMessage()*. See: setErrorMessage, page 5-15.

The text string you enter in the Parameters field must be a valid name of an Oracle Workflow error message. The names of the error messages provided by Oracle Workflow are stored in the NAME column of the WF_RESOURCES table for messages with a type of WFERR.

You can use *Warning()* as a subscription rule function if you want to send the system administrator a warning notification with one of the predefined Workflow error messages whenever a particular event is raised.

For example, define a subscription to the relevant event with the rule function *WF_RULE.Warning* and enter `WFSQL_ARGS` in the Parameters field. Then raise the event to trigger the subscription. Because *Warning()* returns the status code `WARNING`, the Event Manager places the event message on the WF_ERROR queue, but subscription processing for the event still continues. When the listener runs on the WF_ERROR queue, a warning notification will be sent to the system administrator with the message `"Invalid value(s) passed for arguments"`, which is the display name of the `WFSQL_ARGS` error message.

> **Note:** *Warning()* does not raise any exception to the calling application when it completes normally.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Success

PL/SQL Syntax

```
function Success
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Returns the status code `SUCCESS`. This function removes the event message from the queue but executes no other code except returning the `SUCCESS` status code to the calling subscription.

You can use *Success* for testing and debugging purposes while developing code for use with the Business Event System. For example, if you are trying to debug multiple subscriptions to the same event, you can modify one of the subscriptions by replacing its rule function with *WF_RULE.Success*, leaving all other details for the subscription intact. When the subscription is executed, it will return `SUCCESS` but not perform any other subscription processing. This strategy can help you isolate a problem subscription.

Success() is analogous to the WF_STANDARD.Noop procedure used in the standard Noop activity.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Workflow_Protocol

PL/SQL Syntax

```
function Workflow_Protocol
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Sends the event message to the workflow process specified in the subscription, which will in turn send the event message to the inbound agent specified in the subscription.

> **Note:** *Workflow_Protocol()* does not itself send the event message to the inbound agent. This function only sends the event message to the workflow process, where you can model the processing that you want to send the event message on to the specified agent.

If the subscription also specifies an outbound agent, the workflow process places the event message on that agent's queue for propagation to the inbound agent. Otherwise, a default outbound agent will be selected.

If the subscription parameters include the parameter name and value pair `ACKREQ=Y`, then the workflow process waits to receive an acknowledgement after sending the event message.

If the workflow process raises an exception, *Workflow_Protocol()* stores the error information in the event message and returns the status code `ERROR`. Otherwise, *Workflow_Protocol()* returns the status code `SUCCESS`.

Workflow_Protocol() is used as the rule function in several predefined subscriptions to Workflow Send Protocol and Event System Demonstration events. See: Workflow Send Protocol, *Oracle Workflow Developer's Guide* and Event System Demonstration, *Oracle Workflow Developer's Guide*.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Error_Rule

PL/SQL Syntax

```
function Error_Rule
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Performs the same subscription processing as *Default_Rule()*, including:

- Sending the event message to a workflow process, if specified in the subscription definition
- Sending the event message to an agent, if specified in the subscription definition

However, if either of these operations encounters an exception, *Error_Rule()* reraises the exception so that the event is not placed back onto the WF_ERROR queue. Otherwise, *Error_Rule()* returns the status code SUCCESS.

Error_Rule() is used as the rule function for the predefined subscriptions to the Unexpected event and to the Any event with the Error source type. The predefined subscriptions specify that the event should be sent to the Default Event Error process in the System: Error item type.

You can also use this rule function with your own error subscriptions. Enter *WF_RULE.Error* as the rule function for your error subscription and specify the workflow item type and process that you want the subscription to launch.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Related Topics

Unexpected Event, *Oracle Workflow Developer's Guide*

Any Event, *Oracle Workflow Developer's Guide*

SetParametersIntoParameterList

PL/SQL Syntax

```
function SetParametersIntoParameterList
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Sets the parameter name and value pairs from the subscription parameters into the PARAMETER_LIST attribute of the event message, except for any parameter named ITEMKEY or CORRELATION_ID. For a parameter with one of these names, the function sets the CORRELATION_ID attribute of the event message to the parameter value.

If these operations raise an exception, *SetParametersIntoParameterList*() stores the error information in the event message and returns the status code ERROR. Otherwise, *SetParametersIntoParameterList*() returns the status code SUCCESS.

You can use *SetParametersIntoParameterList*() as the rule function for a subscription with a lower phase number, to add predefined parameters from the subscription into the event message. Then subsequent subscriptions with higher phase numbers can access those parameters within the event message.

> **Note:** If the event message will later be sent to a workflow process, then the value for any ITEMKEY or CORRELATION_ID parameter can only

contain single-byte characters, because the CORRELATION_ID attribute of the event message will be used as the item key for the process. The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Related Topics

Event Message Structure, page 5-6

Default_Rule2

PL/SQL Syntax

```
function Default_Rule2
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Performs the default subscription processing only if the PARAMETER_LIST attribute of the event message includes parameters whose names and values match all the parameters defined for the subscription. If the event includes the required parameters, then the rule function calls *Default_Rule()* to perform the following processing:

- Sending the event message to a workflow process, if specified in the subscription definition
- Sending the event message to an agent, if specified in the subscription definition

If either of these operations raises an exception, *Default_Rule2()* traps the exception, stores the error information in the event message, and returns the status code ERROR. Otherwise, *Default_Rule2()* returns the status code SUCCESS.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Related Topics

Default_Rule, page 5-33

Default_Rule3

PL/SQL Syntax

```
function Default_Rule3
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Sets the parameter name and value pairs from the subscription parameters into the PARAMETER_LIST attribute of the event message, and then performs the default subscription processing with the modified event message. This rule function first calls *SetParametersIntoParameterList()* to set the parameters and then calls *Default_Rule()* to perform the following processing:

- Sending the event message to a workflow process, if specified in the subscription definition
- Sending the event message to an agent, if specified in the subscription definition

If either of these operations raises an exception, *Default_Rule3()* traps the exception, stores the error information in the event message, and returns the status code ERROR. Otherwise, *Default_Rule3()* returns the status code SUCCESS.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Related Topics

SetParametersIntoParameterList, page 5-39

Default_Rule, page 5-33

SendNotification

PL/SQL Syntax

```
function SendNotification
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Sends a notification as specified by the parameters in the PARAMETER_LIST attribute of the event message. Use this rule function to send notifications outside of a workflow process.

After sending the notification, this function sets the notification ID into the event parameter list as a parameter named #NID. If you want to use the notification ID in further processing, raise the event using *WF_EVENT.Raise3()*, which returns the event parameter list after Oracle Workflow completes subscription processing for the

event. You can then call *WF_EVENT.GetValueForParameter()* to obtain the value of the `#NID` parameter.

For example, if the notification requires a response, you can retrieve the response values from the user's reply by obtaining the notification ID and using it to call *WF_NOTIFICATION.GetAttrText()*, *WF_NOTIFICATION.GetAttrNumber()*, or *WF_NOTIFICATION.GetAttrDate()* for the RESPOND attributes.

SendNotification() calls the *WF_NOTIFICATION.Send()* API to send the notification, using the event parameters as the input arguments for *WF_NOTIFICATION.Send()*. The following table shows the names of the parameters you should include in the event parameter list to specify the notification you want to send, and the information you should provide in each parameter's value.

Parameters for Sending a Notification

Parameter Name	Parameter Value
RECIPIENT_ROLE	The role name assigned to receive the notification.
MESSAGE_TYPE	The item type associated with the message.
MESSAGE_NAME	The message internal name.
CALLBACK	The callback function name used for communication of SEND and RESPOND source message attributes.
CONTEXT	Context information passed to the callback function.
SEND_COMMENT	A comment presented with the message.
PRIORITY	The priority of the message. If this value is null, the Notification System uses the default priority of the message.
DUE_DATE	The date that a response is required. This optional due date is only for the recipient's information; it has no effect on processing.

Note: Although you can send a notification using the *SendNotification()* rule function without defining or running a workflow process, you do need to define the message you want to send within a workflow item type.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Related Topics

Send, page 4-16

Event Message Structure, page 5-6

Raise3, page 5-24

GetValueForParameter, page 5-31

GetAttribute, page 4-35

Instance_Default_Rule

PL/SQL Syntax

```
function Instance_Default_Rule
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Sends the event to all existing workflow process instances that have eligible receive event activities waiting to receive it. This rule function lets you use a business key attribute to identify one or more workflow processes that should receive the event, instead of sending the event to one particular process based on a specific item type, process name, and item key, as with *Default_Rule()*.

> **Note:** *Instance_Default_Rule()* only sends the event to continue existing workflow processes. If you want to send the event to launch a new process instance, use *Default_Rule()* instead.

First, *Instance_Default_Rule()* calls *SetParametersIntoParameterList()* to set any parameter name and value pairs from the subscription parameters into the `PARAMETER_LIST` attribute of the event message.

Next, the function searches for existing workflow processes that are eligible to receive this event. To be eligible, a workflow process must meet the following requirements:

- The process includes a receive event activity with an activity status of `NOTIFIED`, meaning the process has transitioned to that activity and is waiting to receive the event.
- The event filter for the receive event activity is set to one of the following values:
  - This individual event
  - An event group of which this event is a member
  - NULL, meaning the activity can receive any event
- The receive event activity has an activity attribute named `#BUSINESS_KEY` whose default value is an item type attribute.
- The current value of that item type attribute matches the event key.

After sending the event to all eligible workflow processes, *Instance_Default_Rule()* also sends the event message to an agent, if specified in the subscription definition.

If any operations raise an exception, *Instance_Default_Rule()* traps the exception, stores the error information in the event message, and returns the status code `ERROR`. Otherwise, *Instance_Default_Rule()* returns the status code `SUCCESS`.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Related Topics

SetParametersIntoParameterList, page 5-39

Default_Rule, page 5-33

Event Function APIs

The Event Function APIs provide utility functions that can be called by an application program, the Event Manager, or a workflow process in the runtime phase to communicate with the Business Event System and manage events. These APIs are defined in a PL/SQL package called WF_EVENT_FUNCTIONS_PKG.

- Parameters, page 5-44
- SubscriptionParameters, page 5-45
- AddCorrelation, page 5-46
- Generate, page 5-47
- Receive, page 5-49

Parameters

PL/SQL Syntax

```
function Parameters
  (p_string in varchar2,
   p_numvalues in number,
   p_separator in varchar2) return t_parameters;
```

Description

Parses a string of text that contains the specified number of parameters delimited by the specified separator. *Parameters()* returns the parsed parameters in a varray using the `T_PARAMETERS` composite datatype, which is defined in the WF_EVENT_FUNCTIONS_PKG package. The following table describes the `T_PARAMETERS` datatype:

T_PARAMETERS Datatype

Datatype Name	Element Datatype Definition
T_PARAMETERS	VARCHAR2(240)

Parameters() is a generic utility that you can call in generate functions when the event key is a concatenation of values separated by a known character. Use this function to separate the event key into its component values.

Arguments (input)

p_string
A text string containing concatenated parameters.

p_numvalues
The number of parameters contained in the string.

p_separator
The separator used to delimit the parameters in the string.

Example

Example

```
set serveroutput on

declare
l_parameters wf_event_functions_pkg.t_parameters;
begin
-- Initialize the datatype
l_parameters := wf_event_functions_pkg.t_parameters(1,2);

l_parameters := wf_event_functions_pkg.parameters('1111/2222',2,'/
');
dbms_output.put_line('Value 1:'||l_parameters(1));
dbms_output.put_line('Value 2:'||l_parameters(2));
end;
/
```

SubscriptionParameters

PL/SQL Syntax

```
function SubscriptionParameters
  (p_string in varchar2,
   p_key in varchar2) return varchar2;
```

Description

Returns the value for the specified parameter from a text string containing the parameters defined for an event subscription. The parameter name and value pairs in the text string should be separated by spaces and should appear in the following format:

<name1>=<value1> <name2>=<value2> ... <nameN>=<valueN>

SubscriptionParameters() searches the text string for the specified parameter name and returns the value assigned to that name. For instance, you can call this function in a subscription rule function to retrieve the value of a subscription parameter, and then code different behavior for the rule function based on that value.

Arguments (input)

p_string
A text string containing the parameters defined for an event subscription.

p_key
The name of the parameter whose value should be returned.

Example

Example

In the following example, *SubscriptionParameters()* is used to assign the value of the ITEMKEY subscription parameter to the l_function program variable. The example code is from the *AddCorrelation* function, which adds a correlation ID to an event message during subscription processing. See: AddCorrelation, page 5-46.

```
...
--
-- This is where we will do some logic to determine
-- if there is a parameter
--
  l_function := wf_event_functions_pkg.SubscriptionParameters
    (l_parameters,'ITEMKEY');
...
```

AddCorrelation

PL/SQL Syntax

```
function AddCorrelation
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Adds a correlation ID to an event message during subscription processing. *AddCorrelation()* searches the subscription parameters for a parameter named ITEMKEY that specifies a custom function to generate a correlation ID for the event message. In standalone Oracle Workflow, the function must be specified in the Parameters field for the subscription in the following format:

ITEMKEY=*<package_name.function_name>*

In Oracle Applications, enter ITEMKEY as the parameter name and *<package_name.function_name>* as the parameter value.

AddCorrelation() uses *SubscriptionParameters()* to search for and retrieve the value of the ITEMKEY parameter. See: SubscriptionParameters, page 5-45.

If a custom correlation ID function is specified with the ITEMKEY parameter, then *AddCorrelation()* runs that function and sets the correlation ID to the value returned by the function. Otherwise, *AddCorrelation()* sets the correlation ID to the system date. If the event message is then sent to a workflow process, the Workflow Engine uses that correlation ID as the item key to identify the process instance.

> **Note:** The item key for a process instance can only contain single-byte characters. It cannot contain a multibyte value.

If *AddCorrelation()* encounters an exception, the function returns the status code ERROR. Otherwise, *AddCorrelation()* returns the status code SUCCESS.

AddCorrelation() is defined according the standard API for an event subscription rule function. You can use *AddCorrelation()* as the rule function for a subscription with a low phase number to add a correlation ID to an event, and then use a subscription with a higher phase number to perform any further processing.

For example, follow these steps:

1. Define a subscription to the relevant event with the rule function *WF_EVENT_FUNCTIONS_PKG.AddCorrelation* and a phase of 10. Enter the parameter name and value pair `ITEMKEY=<package_name.function_name>` in the Parameters field for the subscription, replacing `<package_name.function_name>` with the package and function that will generate the correlation ID.
2. Define another subscription to the event with a phase of 20, and specify the processing you want to perform by entering a custom rule function or a workflow item type and process, or both.
3. Raise the event to trigger the subscriptions. The subscription with the lower phase number will be executed first and will add a correlation ID to the event message. When the event is passed to the second subscription, that correlation ID will be used as the item key.

You can also call *AddCorrelation()* within a custom rule function to add a correlation ID during your custom processing. See: Standard API for an Event Subscription Rule Function, *Oracle Workflow Developer's Guide*.

> **Note:** You may find it advantageous to define multiple subscriptions to an event with simple rule functions that you can reuse, rather than creating complex specialized rule functions that cannot be reused.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Generate

PL/SQL Syntax

```
function Generate
  (p_event_name in varchar2,
   p_event_key in varchar2) return clob;
```

Description

Generates the event data for events in the Seed event group. This event data contains Business Event System object definitions which can be used to replicate the objects from one system to another.

The Seed event group includes the following events:

- `oracle.apps.wf.event.event.create`
- `oracle.apps.wf.event.event.update`
- `oracle.apps.wf.event.event.delete`
- `oracle.apps.wf.event.group.create`
- `oracle.apps.wf.event.group.update`

- oracle.apps.wf.event.group.delete
- oracle.apps.wf.event.system.create
- oracle.apps.wf.event.system.update
- oracle.apps.wf.event.system.delete
- oracle.apps.wf.event.agent.create
- oracle.apps.wf.event.agent.update
- oracle.apps.wf.event.agent.delete
- oracle.apps.wf.agent.group.create
- oracle.apps.wf.agent.group.update
- oracle.apps.wf.agent.group.delete
- oracle.apps.wf.event.subscription.create
- oracle.apps.wf.event.subscription.update
- oracle.apps.wf.event.subscription.delete
- oracle.apps.wf.event.all.sync

For the event, event group, system, agent, agent group member, and subscription definition events, *WF_EVENT_FUNCTIONS_PKG.Generate()* calls the Generate APIs associated with the corresponding tables to produce the event data XML document. For the Synchronize Event Systems event, *WF_EVENT_FUNCTIONS_PKG.Generate()* produces an XML document containing all the event, event group, system, agent, agent group member, and subscription definitions from the Event Manager on the local system.

> **Note:** Agent groups are currently only available for the version of Oracle Workflow embedded in Oracle Applications.

Arguments (input)

p_event_name
The internal name of the event.

p_event_key
A string generated when the event occurs within a program or application. The event key uniquely identifies a specific instance of the event.

Related Topics

WF_EVENTS_PKG.Generate, page 5-52

WF_EVENT_GROUPS_PKG.Generate, page 5-53

WF_SYSTEMS_PKG.Generate, page 5-54

WF_AGENTS_PKG.Generate, page 5-55

WF_AGENT_GROUPS_PKG.Generate, page 5-56

WF_EVENT_SUBSCRIPTIONS_PKG.Generate, page 5-58

Predefined Workflow Events, *Oracle Workflow Developer's Guide*

Receive

PL/SQL Syntax

```
function Receive
  (p_subscription_guid in raw,
   p_event in out wf_event_t) return varchar2;
```

Description

Receives Business Event System object definitions during subscription processing and loads the definitions into the appropriate Business Event System tables. This function completes the replication of the objects from one system to another.

WF_EVENT_FUNCTIONS_PKG.Receive() is defined according the the standard API for an event subscription rule function. Oracle Workflow uses *WF_EVENT_FUNCTIONS_PKG.Receive()* as the rule function for two predefined subscriptions, one that is triggered when the System Signup event is raised locally, and one that is triggered when any of the events in the Seed event group is received from an external source.

The Seed event group includes the following events:

- `oracle.apps.wf.event.event.create`
- `oracle.apps.wf.event.event.update`
- `oracle.apps.wf.event.event.delete`
- `oracle.apps.wf.event.group.create`
- `oracle.apps.wf.event.group.update`
- `oracle.apps.wf.event.group.delete`
- `oracle.apps.wf.event.system.create`
- `oracle.apps.wf.event.system.update`
- `oracle.apps.wf.event.system.delete`
- `oracle.apps.wf.event.agent.create`
- `oracle.apps.wf.event.agent.update`
- `oracle.apps.wf.event.agent.delete`
- `oracle.apps.wf.agent.group.create`
- `oracle.apps.wf.agent.group.update`
- `oracle.apps.wf.agent.group.delete`
- `oracle.apps.wf.event.subscription.create`
- `oracle.apps.wf.event.subscription.update`
- `oracle.apps.wf.event.subscription.delete`
- `oracle.apps.wf.event.all.sync`

WF_EVENT_FUNCTIONS_PKG.Receive() parses the event data XML document from the event message that was received and then loads the Business Event System object definitions into the appropriate tables.

Note: For the event, event group, system, agent, agent group, and subscription definition events, *WF_EVENT_FUNCTIONS_PKG.Receive()* calls the Receive APIs associated with the corresponding tables to parse the XML document and load the definition into the table.

Agent groups are currently only available for the version of Oracle Workflow embedded in Oracle Applications.

Arguments (input)

p_subscription_guid
The globally unique identifier of the subscription.

p_event
The event message.

Related Topics

WF_EVENTS_PKG.Receive, page 5-52

WF_EVENT_GROUPS_PKG.Receive, page 5-53

WF_SYSTEMS_PKG.Receive, page 5-54

WF_AGENTS_PKG.Receive, page 5-55

WF_AGENT_GROUPS_PKG.Receive, page 5-57

WF_EVENT_SUBSCRIPTIONS_PKG.Receive, page 5-58

Predefined Workflow Events, *Oracle Workflow Developer's Guide*

Business Event System Replication APIs

You can call the following APIs to replicate Business Event System data across your systems. The replication APIs are stored in the following PL/SQL packages, each of which corresponds to a Business Event System table. Oracle Workflow provides both a generate function and a receive function for each table.

- WF_EVENTS_PKG
  - WF_EVENTS_PKG.Generate, page 5-52
  - WF_EVENTS_PKG.Receive, page 5-52
- WF_EVENT_GROUPS_PKG
  - WF_EVENT_GROUPS_PKG.Generate, page 5-53
  - WF_EVENT_GROUPS_PKG.Receive, page 5-53
- WF_SYSTEMS_PKG
  - WF_SYSTEMS_PKG.Generate, page 5-54
  - WF_SYSTEMS_PKG.Receive, page 5-54
- WF_AGENTS_PKG
  - WF_AGENTS_PKG.Generate, page 5-55
  - WF_AGENTS_PKG.Receive, page 5-55
- WF_AGENT_GROUPS_PKG

    - WF_AGENT_GROUPS_PKG.Generate, page 5-56
    - WF_AGENT_GROUPS_PKG.Receive, page 5-57
- WF_EVENT_SUBSCRIPTIONS_PKG
    - WF_EVENT_SUBSCRIPTIONS_PKG.Generate, page 5-58
    - WF_EVENT_SUBSCRIPTIONS_PKG.Receive, page 5-58

    Each generate API produces an XML message containing the complete information from the appropriate table for the specified Business Event System object definition. The corresponding receive API parses the XML message and loads the row into the appropriate table.

Oracle Workflow uses these APIs during the automated replication of Business Event System data. The generate APIs are called by *WF_EVENT_FUNCTIONS_PKG.Generate()*, while the receive APIs are called by *WF_EVENT_FUNCTIONS_PKG.Receive()*. See: Generate, page 5-47 and Receive, page 5-49.

Document Type Definitions

The document type definitions (DTDs) for the Workflow table XML messages are defined under the master tag `WF_TABLE_DATA`. Beneath the master tag, each DTD has a tag identifying the Workflow table name to which it applies, and beneath that, a version tag as well as tags for each column in the table. The following example shows how the DTDs are structured:

```
<WF_TABLE_DATA>              <- masterTagName
  <WF_TABLE_NAME>            <- m_table_name
    <VERSION></VERSION>      <- m_package_version
    <COL1></COL1>
    <COL2></COL2>
  </WF_TABLE_NAME>
</WF_TABLE_DATA>
```

The Business Event System replication APIs use the following DTDs:

- WF_EVENTS DTD, page 5-51
- WF_EVENT_GROUPS DTD, page 5-53
- WF_SYSTEMS DTD, page 5-54
- WF_AGENTS DTD, page 5-55
- WF_AGENT_GROUPS DTD, page 5-56

    Note: Agent groups are currently only available for the version of Oracle Workflow embedded in Oracle Applications.

- WF_EVENT_SUBSCRIPTIONS DTD, page 5-57

WF_EVENTS Document Type Definition

The following document type definition (DTD) describes the required structure for an XML message that contains the complete information for an event definition in the WF_EVENTS table.

```
<WF_TABLE_DATA>
  <WF_EVENTS>
    <VERSION></VERSION>
    <GUID></GUID>
    <NAME></NAME>
    <TYPE></TYPE>
    <STATUS></STATUS>
    <GENERATE_FUNCTION></GENERATE_FUNCTION>
    <OWNER_NAME></OWNER_NAME>
    <OWNER_TAG></OWNER_TAG>
    <CUSTOMIZATION_LEVEL></CUSTOMIZATION_LEVEL>
    <LICENSED_FLAG></LICENSED_FLAG>
    <DISPLAY_NAME></DISPLAY_NAME>
    <DESCRIPTION></DESCRIPTION>
  </WF_EVENTS>
</WF_TABLE_DATA>
```

WF_EVENTS_PKG.Generate

PL/SQL Syntax

```
function Generate
  (x_guid in raw)
  return varchar2;
```

Description

Generates an XML message containing the complete information from the WF_EVENTS table for the specified event definition.

Arguments (input)

x_guid
The globally unique identifier of the event.

WF_EVENTS_PKG.Receive

PL/SQL Syntax

```
procedure Receive
  (x_message in varchar2);
```

Description

Receives an XML message containing the complete information for an event definition and loads the information into the WF_EVENTS table.

Arguments (input)

x_message
An XML message containing the complete information for an event definition.

WF_EVENT_GROUPS Document Type Definition

The following document type definition (DTD) describes the required structure for an XML message that contains the complete information for an event group member definition in the WF_EVENT_GROUPS table.

> **Note:** Event group header information is defined in the WF_EVENTS table, similarly to an individual event. Only the event group member definitions are stored in the WF_EVENT_GROUPS table.

```
<WF_TABLE_DATA>
  <WF_EVENT_GROUPS>
    <VERSION></VERSION>
    <GROUP_GUID></GROUP_GUID>
    <MEMBER_GUID></MEMBER_GUID>
  </WF_EVENT_GROUPS>
</WF_TABLE_DATA>
```

WF_EVENT_GROUPS_PKG.Generate

PL/SQL Syntax

```
function Generate
  (x_group_guid in raw,
   x_member_guid in raw)
  return varchar2;
```

Description

Generates an XML message containing the complete information from the WF_EVENT_GROUPS table for the specified event group member definition.

Arguments (input)

x_group_guid
The globally unique identifier of the event group.

x_member_guid
The globally unique identifier of the individual member event.

WF_EVENT_GROUPS_PKG.Receive

PL/SQL Syntax

```
procedure Receive
  (x_message in varchar2);
```

Description

Receives an XML message containing the complete information for an event group member definition and loads the information into the WF_EVENT_GROUPS table.

Arguments (input)

x_message
An XML message containing the complete information for an event group member definition.

WF_SYSTEMS Document Type Definition

The following document type definition (DTD) describes the required structure for an XML message that contains the complete information for a system definition in the WF_SYSTEMS table.

```
<WF_TABLE_DATA>
  <WF_SYSTEMS>
    <VERSION></VERSION>
    <GUID></GUID>
    <NAME></NAME>
    <MASTER_GUID></MASTER_GUID>
    <DISPLAY_NAME></DISPLAY_NAME>
    <DESCRIPTION></DESCRIPTION>
  </WF_SYSTEMS>
</WF_TABLE_DATA>
```

WF_SYSTEMS_PKG.Generate

PL/SQL Syntax

```
function Generate
  (x_guid in raw)
  return varchar2;
```

Description

Generates an XML message containing the complete information from the WF_SYSTEMS table for the specified system definition.

Arguments (input)

x_guid
The globally unique identifier of the system.

WF_SYSTEMS_PKG.Receive

PL/SQL Syntax

```
procedure Receive
  (x_message in varchar2);
```

Description

Receives an XML message containing the complete information for a system definition and loads the information into the WF_SYSTEMS table.

Arguments (input)

x_message
An XML message containing the complete information for a system definition.

WF_AGENTS Document Type Definition

The following document type definition (DTD) describes the required structure for an XML message that contains the complete information for an agent definition in the WF_AGENTS table.

```
<WF_TABLE_DATA>
  <WF_AGENTS>
    <VERSION></VERSION>
    <GUID></GUID>
    <NAME></NAME>
    <SYSTEM_GUID></SYSTEM_GUID>
    <PROTOCOL></PROTOCOL>
    <ADDRESS></ADDRESS>
    <QUEUE_HANDLER></QUEUE_HANDLER>
    <QUEUE_NAME></QUEUE_NAME>
    <DIRECTION></DIRECTION>
    <STATUS></STATUS>
    <DISPLAY_NAME></DISPLAY_NAME>
    <DESCRIPTION></DESCRIPTION>
    <TYPE></TYPE>
  </WF_AGENTS>
</WF_TABLE_DATA>
```

WF_AGENTS_PKG.Generate

PL/SQL Syntax

```
function Generate
  (x_guid in raw)
  return varchar2;
```

Description

Generates an XML message containing the complete information from the WF_AGENTS table for the specified agent definition.

Arguments (input)

x_guid
The globally unique identifier of the agent.

WF_AGENTS_PKG.Receive

PL/SQL Syntax

```
procedure Receive
  (x_message in varchar2);
```

Description

Receives an XML message containing the complete information for an agent definition and loads the information into the WF_AGENTS table.

Arguments (input)

x_message
An XML message containing the complete information for an agent definition.

WF_AGENT_GROUPS Document Type Definition

The following document type definition (DTD) describes the required structure for an XML message that contains the complete information for an agent group member definition in the WF_AGENT_GROUPS table.

> **Note:** Agent group header information is defined in the WF_AGENTS table, similarly to an individual agent. Only the agent group member definitions are stored in the WF_AGENT_GROUPS table.
>
> Agent groups are currently only available for the version of Oracle Workflow embedded in Oracle Applications.

```
<WF_TABLE_DATA>
  <WF_AGENT_GROUPS>
    <VERSION></VERSION>
    <GROUP_GUID></GROUP_GUID>
    <MEMBER_GUID></MEMBER_GUID>
  </WF_AGENT_GROUPS>
</WF_TABLE_DATA>
```

WF_AGENT_GROUPS_PKG.Generate

PL/SQL Syntax

```
function Generate
  (x_group_guid in raw,
   x_member_guid in raw)
  return varchar2;
```

Description

Generates an XML message containing the complete information from the WF_AGENT_GROUPS table for the specified agent group member definition.

> **Note:** Agent groups are currently only available for the version of Oracle Workflow embedded in Oracle Applications.

Arguments (input)

x_group_guid
The globally unique identifier of the agent group.

x_member_guid
The globally unique identifier of the individual member agent.

WF_AGENT_GROUPS_PKG.Receive

PL/SQL Syntax

```
procedure Receive
  (x_message in varchar2);
```

Description

Receives an XML message containing the complete information for an agent group member definition and loads the information into the WF_AGENT_GROUPS table.

> **Note:** Agent groups are currently only available for the version of Oracle Workflow embedded in Oracle Applications.

Arguments (input)

x_message
An XML message containing the complete information for an agent group member definition.

WF_EVENT_SUBSCRIPTIONS Document Type Definition

The following document type definition (DTD) describes the required structure for an XML message that contains the complete information for an event subscription definition in the WF_EVENT_SUBSCRIPTIONS table.

```
<WF_TABLE_DATA>
  <WF_EVENT_SUBSCRIPTIONS>
    <VERSION></VERSION>
    <GUID></GUID>
    <SYSTEM_GUID></SYSTEM_GUID>
    <SOURCE_TYPE></SOURCE_TYPE>
    <SOURCE_AGENT_GUID></SOURCE_AGENT_GUID>
    <EVENT_FILTER_GUID></EVENT_FILTER_GUID>
    <PHASE></PHASE>
    <STATUS></STATUS>
    <RULE_DATA></RULE_DATA>
    <OUT_AGENT_GUID></OUT_AGENT_GUID>
    <TO_AGENT_GUID></TO_AGENT_GUID>
    <PRIORITY></PRIORITY>
    <RULE_FUNCTION></RULE_FUNCTION>
    <WF_PROCESS_TYPE></WF_PROCESS_TYPE>
    <WF_PROCESS_NAME></WF_PROCESS_NAME>
    <PARAMETERS></PARAMETERS>
    <OWNER_NAME></OWNER_NAME>
    <OWNER_TAG></OWNER_TAG>
    <CUSTOMIZATION_LEVEL></CUSTOMIZATION_LEVEL>
    <LICENSED_FLAG></LICENSED_FLAG>
    <DESCRIPTION></DESCRIPTION>
    <EXPRESSION></EXPRESSION>
  </WF_EVENT_SUBSCRIPTIONS>
</WF_TABLE_DATA>
```

WF_EVENT_SUBSCRIPTIONS_PKG.Generate

PL/SQL Syntax

```
function Generate
  (x_guid in raw)
  return varchar2;
```

Description

Generates an XML message containing the complete information from the WF_EVENT_SUBSCRIPTIONS table for the specified event subscription definition.

Arguments (input)

x_guid
The globally unique identifier of the event subscription.

WF_EVENT_SUBSCRIPTIONS_PKG.Receive

PL/SQL Syntax

```
procedure Receive
  (x_message in varchar2);
```

Description

Receives an XML message containing the complete information for an event subscription definition and loads the information into the WF_EVENT_SUBSCRIPTIONS table.

Arguments (input)

x_message
An XML message containing the complete information for an event subscription definition.

Business Event System Cleanup API

The Workflow Business Event System cleanup API can be used to clean up the standard WF_CONTROL queue in the Business Event System by removing inactive subscribers from the queue. This API is defined in a PL/SQL package called WF_BES_CLEANUP.

Cleanup_Subscribers

PL/SQL Syntax

```
procedure Cleanup_Subscribers
  (errbuf out varchar2,
   retcode out varchar2);
```

Description

Performs cleanup for the standard WF_CONTROL queue.

When a middle tier process for Oracle Applications or for standalone Oracle Workflow starts up, it creates a JMS subscriber to the WF_CONTROL queue. Then, when an event message is placed on the queue, a copy of the event message is created for each subscriber to the queue. If a middle tier process dies, however, the corresponding subscriber remains in the database. For more efficient processing, you should ensure that WF_CONTROL is periodically cleaned up by running *Cleanup_Subscribers()* to remove the subscribers for any middle tier processes that are no longer active.

The *Cleanup_Subscribers()* procedure sends an event named `oracle.apps.wf.bes.control.ping` to check the status of each subscriber to the WF_CONTROL queue. If the corresponding middle tier process is still alive, it sends back a response.

The next time the cleanup procedure runs, it checks whether responses have been received for each ping event sent during the previous run. If no response was received from a particular subscriber, that subscriber is removed.

Finally, after removing any subscribers that are no longer active, the procedure sends a new ping event to the remaining subscribers.

The recommended frequency for performing cleanup is every twelve hours. In order to allow enough time for subscribers to respond to the ping event, the minimum wait time between two cleanup runs is thirty minutes. If you run the procedure again less than thirty minutes after the last run, it will not perform any processing.

The maximum retention time for information about ping events sent to subscribers is thirty days. *Cleanup_Subscribers()* deletes information for previously sent pings that are more than thirty days old.

The procedure returns an error buffer that contains an error message if any inactive subscriber could not be removed during the cleanup. It also returns one of the following codes to indicate the status of the cleanup.

- `0` - Success
- `1` - Warning
- `2` - Error

Related Topics

Cleaning Up the Workflow Control Queue, *Oracle Workflow Administrator's Guide*

Standard Agents, *Oracle Workflow Developer's Guide*

Business Event System Control Events, *Oracle Workflow Developer's Guide*

6

Workflow Queue APIs

This chapter describes the APIs for Oracle Workflow Advanced Queues processing. The APIs consist of PL/SQL functions and procedures to handle workflow Advanced Queues processing. Although these APIs will continue to be supported for backward compatibility, customers using Oracle Workflow Release 2.6 and higher should use the Business Event System rather than the queue APIs to integrate with Oracle Advanced Queuing.

This chapter covers the following topics:

- Workflow Queue APIs

Workflow Queue APIs

Oracle Workflow queue APIs can be called by an application program or a workflow function in the runtime phase to handle workflow Advanced Queues processing.

> **Note:** Although these APIs will continue to be supported for backward compatibility, customers using Oracle Workflow Release 2.6 and higher should use the Business Event System rather than the queue APIs to integrate with Oracle Advanced Queuing.
>
> In a future release, this workflow Advanced Queues processing will be implemented within the Business Event System using a specialized queue handler to handle dequeue and enqueue operations.

In Oracle Workflow, an 'outbound' and an 'inbound' queue are established. A package of data on the queue is referred to as an event or a message.

> **Note:** An event in this context is different from the business events associated with the Business Event System, and a message in this context is different from the messages associated with notification activities.

Events are enqueued in the outbound queue for agents to consume and process. These agents may be any application that is external to the database. Similarly an agent may enqueue some message to the inbound queue for the Workflow Engine to consume and process. The outbound and inbound queues facilitate the integration of external activities into your workflow processes.

> **Note:** Background engines use a separate 'deferred' queue.

All Oracle Workflow queue APIs are defined in a PL/SQL package called WF_QUEUE. You must execute these queue APIs from the same Oracle Workflow account since the APIs are account dependent.

> **Important:** In using these APIs, we assume that you have prior knowledge of Oracle Advanced Queuing concepts and terminology. Refer to the *Oracle Application Developer's Guide - Advanced Queuing* or *Oracle Streams Advanced Queuing User's Guide and Reference* for more information on Advanced Queues.

Queue APIs

Developer APIs for the Inbound Queue

The following APIs are for developers who wish to write to the inbound queue by creating messages in the internal stack rather than using WF_QUEUE.EnqueueInbound(). The internal stack is purely a storage area and you must eventually write each message that you create on the stack to the inbound queue.

> **Note:** For efficient performance, you should periodically write to the inbound queue to prevent the stack from growing too large.

Payload Structure

Oracle Workflow queues use the datatype `system.wf_payload_t` to define the payload for any given message. The payload contains all the information that is required about the event. The following table lists the attributes of `system.wf_payload_t`.

System.wf_payload_t Attributes

Attribute Name	Datatype	Description
ITEMTYPE	VARCHAR2(8)	The item type of the event.
ITEMKEY	VARCHAR2(240)	The item key of the event.
ACTID	NUMBER	The function activity instance ID.
FUNCTION_NAME	VARCHAR2(200)	The name of the function to execute.
PARAM_LIST	VARCHAR2(4000)	A list of "`value_name=value`" pairs. In the inbound scenario, the pairs are passed as item attributes and item attribute values. In the outbound scenario, the pairs are passed as all the attributes and attribute values of the function (activity attributes).
RESULT	VARCHAR2(30)	An optional activity completion result. Possible values are determined by the function activity's Result Type or can be an engine standard result.

Related Topics

Standard API for PL/SQL Procedures Called by Function Activities, *Oracle Workflow Developer's Guide*

EnqueueInbound

Syntax

```
procedure EnqueueInbound
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number,
   result in varchar2 default null,
   attrlist in varchar2 default null,
   correlation in varchar2 default null,
   error_stack in varchar2 default null);
```

Description

Enqueues the result from an outbound event onto the inbound queue. An outbound event is defined by an outbound queue message that is consumed by some agent.

Oracle Workflow marks the external function activity as complete with the specified result when it processes the inbound queue. The result value is only effective for successful completion, however. If you specify an external program error in the `error_stack` parameter, Oracle Workflow marks the external function activity

as complete with an ERROR status, overriding the result value. Additionally, if a corresponding error process is defined in the item type, Oracle Workflow launches that error process.

Arguments (input)

itemtype
The item type of the event.

itemkey
The item key of the event. An item key is a string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process instance.

actid
The function activity instance ID that this event is associated with.

result
An optional activity completion result. Possible values are determined by the function activity's Result Type.

attrlist
A longlist of "value name=value" pairs that you want to pass back as item attributes and item attribute values. Each pair must be delimited by the caret character (^), as in the example, "ATTR1=A^ATTR2=B^ATTR3=C". If a specified value name does not exist as an item attribute, Oracle Workflow creates the item attribute for you, of type varchar2.

correlation
Specify an optional correlation identifier for the message to be enqueued. Oracle Advanced Queues allow you to search a queue for messages based on a specific correlation value. If null, the Workflow Engine creates a correlation identifier based on the Workflow schema name and the item type.

error_stack
Specify an optional external program error that will be placed on Oracle Workflow's internal error stack. You can specify any text value up to a maximum length of 200 characters.

DequeueOutbound

Syntax

```
procedure DequeueOutbound
  (dequeuemode in number,
   navigation in number default 1,
   correlation in varchar2 default null,
   itemtype in varchar2 default null,
   payload out system.wf_payload_t,
   message_handle in out raw,
   timeout out boolean);
```

Description

Dequeues a message from the outbound queue for some agent to consume.

> **Important:** If you call this procedure within a loop, you must remember to set the returned message handle to null, otherwise, the procedure

dequeues the same message again. This may not be the behavior you want and may cause an infinite loop.

Arguments (input)

dequeuemode
A value of DBMS_AQ.BROWSE, DBMS_AQ.LOCKED, or DBMS_AQ.REMOVE, corresponding to the numbers `1`, `2` and `3` respectively, to represent the locking behavior of the dequeue. A mode of DBMS_AQ.BROWSE means to read the message from the queue without acquiring a lock on the message. A mode of DBMS_AQ.LOCKED means to read and obtain a write lock on the message, where the lock lasts for the duration of the transaction. A mode of DBMS_AQ.REMOVE means read the message and delete it.

navigation
Specify DBMS_AQ.FIRST_MESSAGE or DBMS_AQ.NEXT_MESSAGE, corresponding to the number `1` or `2` respectively, to indicate the position of the message that will be retrieved. A value of DBMS_AQ.FIRST_MESSAGE retrieves the first message that is available and matches the correlation criteria. The first message is inherently the beginning of the queue. A value of DBMS_AQ.NEXT_MESSAGE retrieves the next message that is available and matches the correlation criteria, and lets you read through the queue. The default is `1`.

correlation
Specify an optional correlation identifier for the message to be dequeued. Oracle Advanced Queues allow you to search a queue for messages based on a specific correlation value. You can use the Like comparison operator, `'%'`, to specify the identifier string. If null, the Workflow Engine creates a correlation identifier based on the Workflow schema name and the item type.

itemtype
The item type of the event.

message_handle
Specify an optional message handle ID for the specific event to be dequeued. If you specify a message handle ID, the correlation identifier is ignored.

> **Important:** The timeout output returns `TRUE` when there is nothing further to read in the queue.

Example

Example
Following is an example of code that loops through the outbound queue and displays the output.

```
declare

  event           system.wf_payload_t;
  i               number;
  msg_id          raw(16);
  queuename       varchar2(30);
  navigation_mode number;
  end_of_queue    boolean;

begin
  queuename:=wf_queue.OUTBOUNDQUEUE;
  i:=0;
  LOOP
    i:=i+1;

    -- always start with the first message then progress to next
    if i = 1 then
      navigation_mode := dbms_aq.FIRST_MESSAGE;
    else
      navigation_mode := dbms_aq.NEXT_MESSAGE;
    end if;

    -- not interested in specific msg_id. Leave it null so
    -- as to loop through all messages in queue
    msg_id :=null;

    wf_queue.DequeueOutbound(
      dequeuemode     => dbms_aq.BROWSE,
      payload         => event,
      navigation      => navigation_mode,
      message_handle  => msg_id,
      timeout         => end_of_queue);

    if end_of_queue then
      exit;
    end if;

    -- print the correlation itemtype:itemKey
    dbms_output.put_line('Msg '||to_char(i)||' = '||
      event.itemtype||':'||event.itemkey||' '||
      event.actid||' '||event.param_list);

  END LOOP;

end;
/
```

DequeueEventDetail

Syntax

```
procedure DequeueEventDetail
  (dequeuemode in number,
   navigation in number default 1,
   correlation in varchar2 default null,
   itemtype in out varchar2,
   itemkey out varchar2,
   actid out number,
   function_name out varchar2,
   param_list out varchar2,
   message_handle in out raw,
   timeout out boolean);
```

Description

Dequeue from the outbound queue, the full event details for a given message. This API is similar to DequeueOutbound except it does not reference the payload type. Instead, it outputs `itemkey`, `actid`, `function_name`, and `param_list`, which are part of the payload.

> **Important:** If you call this procedure within a loop, you must remember to set the returned message handle to null, otherwise, the procedure dequeues the same message again. This may not be the behavior you want and may cause an infinite loop.

Arguments (input)

dequeuemode
A value of DBMS_AQ.BROWSE, DBMS_AQ.LOCKED, or DBMS_AQ.REMOVE, corresponding to the numbers `1`, `2` and `3` respectively, to represent the locking behavior of the dequeue. A mode of DBMS_AQ.BROWSE means to read the message from the queue without acquiring a lock on the message. A mode of DBMS_AQ.LOCKED means to read and obtain a write lock on the message, where the lock lasts for the duration of the transaction. A mode of DBMS_AQ.REMOVE means read the message and update or delete it.

navigation
Specify DBMS_AQ.FIRSTMESSAGE or DBMS_AQ.NEXTMESSAGE, corresponding to the number `1` or `2` respectively, to indicate the position of the message that will be retrieved. A value of DBMS_AQ.FIRSTMESSAGE retrieves the first message that is available and matches the correlation criteria. It also resets the position to the beginning of the queue. A value of DBMS_AQ.NEXTMESSAGE retrieves the next message that is available and matches the correlation criteria. The default is `1`.

correlation
Specify an optional correlation identifier for the message to be dequeued. Oracle Advanced Queues allow you to search a queue for messages based on a specific correlation value. You can use the Like comparison operator, '`%`', to specify the identifier string. If null, the Workflow Engine creates a correlation identifier based on the Workflow schema name and the item type.

acctname
The Oracle Workflow database account name. If `acctname` is null, it defaults to the pseudocolumn `USER`.

itemtype
Specify an optional item type for the message to dequeue if you are not specifying a correlation.

message_handle
Specify an optional message handle ID for the specific event to be dequeued. If you specify a message handle ID, the correlation identifier is ignored.

> **Important:** The timeout output returns `TRUE` when there is nothing further to read in the queue.

PurgeEvent

Syntax

```
procedure PurgeEvent
  (queuename in varchar2,
   message_handle in raw);
```

Description

Removes an event from a specified queue without further processing.

Arguments (input)

queuename
The name of the queue from which to purge the event.

message_handle
The message handle ID for the specific event to purge.

PurgeItemType

Syntax

```
procedure PurgeItemType
  (queuename in varchar2,
   itemtype in varchar2 default null,
   correlation in varchar2 default null);
```

Description

Removes all events belonging to a specific item type from a specified queue without further processing.

Arguments (input)

queuename
The name of the queue from which to purge the events.

itemtype
An optional item type of the events to purge.

correlation
Specify an optional correlation identifier for the message to be purged. Oracle Advanced Queues allow you to search a queue for messages based on a specific correlation value. You can use the Like comparison operator, '%', to specify the identifier string. If null, the Workflow Engine creates a correlation identifier based on the Workflow schema name and the item type.

ProcessInboundQueue

Syntax

```
procedure ProcessInboundQueue
  (itemtype in varchar2 default null,
   correlation in varchar2 default null);
```

Description

Reads every message off the inbound queue and records each message as a completed event. The result of the completed event and the list of item attributes that are updated as a consequence of the completed event are specified by each message in the inbound queue. See: EnqueueInbound, page 6-3.

Arguments (input)

itemtype
An optional item type of the events to process.

correlation
If you wish to process only messages with a specific correlation, enter a correlation identifier. If `correlation` is null, the Workflow Engine creates a correlation identifier based on the Workflow schema name and the item type.

GetMessageHandle

Syntax

```
function GetMessageHandle
  (queuename in varchar2,
   itemtype in varchar2,
   itemkey in varchar2,
   actid in number,
   correlation in varchar2 default null)
   return raw;
```

Description

Returns a message handle ID for a specified message.

Arguments (input)

queuename
The name of the queue from which to retrieve the message handle.

itemtype
The item type of the message.

itemkey
The item key of the message. An item key is a string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process instance.

actid
The function activity instance ID that this message is associated with.

correlation
Specify an optional correlation identifier for the message. If the correlation is null, the Workflow Engine creates a correlation identifier based on the Workflow schema name and the item type.

DequeueException

Syntax

```
procedure DequeueException
  (queuename in varchar2);
```

Description

Dequeues all messages from an exception queue and places the messages on the standard Business Event System WF_ERROR queue with the error message `'Message Expired'`. When the messages are dequeued from WF_ERROR, a predefined subscription is triggered that launches the Default Event Error process.

Arguments (input)

queuename
The name of the exception queue that has been enabled for dequeue.

Related Topics

Default Event Error Process, *Oracle Workflow Developer's Guide*

DeferredQueue

Syntax

```
function DeferredQueue return varchar2;
```

Description

Returns the name of the queue and schema used by the background engine for deferred processing.

InboundQueue

Syntax

```
function InboundQueue return varchar2;
```

Description

Returns the name of the inbound queue and schema. The inbound queue contains messages for the Workflow Engine to consume.

OutboundQueue

Syntax

```
function OutboundQueue return varchar2;
```

Description

Returns the name of the outbound queue and schema. The outbound queue contains messages for external agents to consume.

ClearMsgStack

Syntax

```
procedure ClearMsgStack;
```

Description

Clears the internal stack. See: Developer APIs for the Inbound Queue, page 6-2.

CreateMsg

Syntax

```
procedure CreateMsg
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number);
```

Description

Creates a new message in the internal stack if it doesn't already exist. See: Developer APIs for the Inbound Queue, page 6-2.

Arguments (input)

itemtype
The item type of the message.

itemkey
The item key of the message. An item key is a string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process instance.

actid
The function activity instance ID that this message is associated with.

WriteMsg

Syntax

```
procedure WriteMsg
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number);
```

Description

Writes a message from the internal stack to the inbound queue. See: Developer APIs for the Inbound Queue, page 6-2.

Arguments (input)

itemtype
The item type of the message.

itemkey
The item key of the message. An item key is a string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process.

actid
The function activity instance ID that this message is associated with.

SetMsgAttr

Syntax

```
procedure SetMsgAttr
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number,
   attrName in varchar2,
   attrValue in varchar2);
```

Description

Appends an item attribute to the message in the internal stack. See: Developer APIs for the Inbound Queue, page 6-2.

Arguments (input)

itemtype
The item type of the message.

itemkey
The item key of the message. An item key is a string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process instance.

actid
The function activity instance ID that this message is associated with.

attrName
The internal name of the item attribute you wish to append to the message.

attrValue
The value of the item attribute you wish to append.

SetMsgResult

Syntax

```
procedure SetMsgResult
  (itemtype in varchar2,
   itemkey in varchar2,
   actid in number,
   result in varchar2);
```

Description

Sets a result to the message written in the internal stack. See: Developer APIs for the Inbound Queue, page 6-2.

Arguments (input)

itemtype
The item type of the message.

itemkey
The item key of the message. An item key is a string generated from the application object's primary key. The string uniquely identifies the item within an item type. The item type and key together identify the process instance.

actid
The function activity instance ID that this message is associated with.

result
The completion result for the message. Possible values are determined by the activity's Result Type.

7

Document Management APIs

This chapter describes the APIs for Oracle Workflow document management. The APIs consist of PL/SQL functions and procedures to integrate with document management systems. Document management functionality is reserved for future use. This description of Oracle Workflow document management APIs is provided for reference only.

This chapter covers the following topics:

- Document Management APIs

Document Management APIs

Important: Document management functionality is reserved for future use. This description of Oracle Workflow document management APIs is provided for reference only.

The following document management APIs can be called by user interface (UI) agents to return URLs or javascript functions that enable integrated access to supported document management systems. All supported document management (DM) systems accommodate a URL interface to access documents.

The document management APIs allow you to access documents across multiple instances of the same DM system, as well as across multiple instances of DM systems from different vendors within the same network.

The document management APIs are defined in a PL/SQL package called FND_DOCUMENT_MANAGEMENT.

- get_launch_document_url, page 7-2
- get_launch_attach_url, page 7-2
- get_open_dm_display_window, page 7-3
- get_open_dm_attach_window, page 7-3
- set_document_id_html, page 7-4

Related Topics

Standard API for PL/SQL Procedures Called by Function Activities, *Oracle Workflow Developer's Guide*

get_launch_document_url

Important: Document management functionality is reserved for future use. This description of the get_launch_document_url API is provided for reference only.

Syntax

```
procedure get_launch_document_url
  (username in varchar2,
   document_identifier in varchar2,
   display_icon in Boolean,
   launch_document_url out varchar2);
```

Description

Returns an anchor URL that launches a new browser window containing the DM integration screen that displays the specified document. The screen is a frame set of two frames. The upper frame contains a customizable company logo and a toolbar of Oracle Workflow-integrated document management functions. The lower frame displays the specified document.

Arguments (input)

username
The username of the person accessing the document management system.

document_identifier
The document identifier for the document you wish to display. The document identifier should be stored as a value in an item attribute of type document. You can retrieve the document identifier using the *GetItemAttrDocument* API. See: GetItemAttrDocument, page 2-43 and SetItemAttrDocument, page 2-38.

display_icon
True or False. True tells the procedure to return the URL with the paper clip attachment icon and translated prompt name, whereas False tells the procedure to return only the URL. This argument provides you the flexibility needed when you call this procedure from a form- or HTML-based UI agent.

get_launch_attach_url

Important: Document management functionality is reserved for future use. This description of the get_launch_attach_url API is provided for reference only.

Syntax

```
procedure get_launch_attach_url
  (username in varchar2,
   callback_function in varchar2,
   display_icon in Boolean,
   launch_attach_url out varchar2);
```

Description

Returns an anchor URL that launches a new browser window containing a DM integration screen that allows you to attach a document. The screen is a frame set of two frames. The upper frame contains a customizable company logo and a toolbar of Oracle Workflow-integrated document management functions. The lower frame displays the search screen of the default document management system.

Arguments (input)

username
The username of the person accessing the document management system.

callback_function
The URL you would like to invoke after the user selects a document to attach. This callback function should be the callback_url syntax that is returned from the *set_document_id_html* API.

display_icon
True or False. True tells the procedure to return the URL with the paper clip attachment icon and translated prompt name, whereas False tells the procedure to return only the URL. This argument provides you the flexibility needed when you call this procedure from a form- or HTML-based UI agent.

get_open_dm_display_window

Important: Document management functionality is reserved for future use. This description of the get_open_dm_display_window API is provided for reference only.

Syntax

```
procedure get_open_dm_display_window
```

Description

Returns a javascript function that displays an attached document from the current UI. The javascript function is used by all the document management functions that the user can perform on an attached document. Each DM function also gives the current DM integration screen a name so that the Document Transport Window can call back to the javascript function in the current window.

get_open_dm_attach_window

Important: Document management functionality is reserved for future use. This description of the get_open_dm_attach_window API is provided for reference only.

Syntax

```
procedure get_open_dm_attach_window
```

Description

Returns a javascript function to open a Document Transport Window when a user tries to attach a document in the current UI. The javascript function is used by all the document management functions that the user can perform to attach a document. Each DM function also gives the current DM integration screen a name so that the Document Transport Window can call back to the javascript function in the current window.

set_document_id_html

Important: Document management functionality is reserved for future use. This description of the set_document_id_html API is provided for reference only.

Syntax

```
procedure set_document_id_html
  (frame_name in varchar2,
   form_name in varchar2,
   document_id_field_name in varchar2
   document_name_field_name in varchar2,
   callback_url out varchar2);
```

Description

Returns a callback URL that gets executed when a user selects a document from the DM system. Use this procedure to set the document that is selected from the document management Search function to the specified destination field of an HTML page. The destination field is the field from which the user launches the DM integration screen to attach a document. Pass the returned callback URL as an argument to the *get_launch_attach_url* API.

Arguments (input)

frame_name
The name of the HTML frame that you wish to interact with in the current UI.

form_name
The name of the HTML form that you wish to interact with in the current UI.

document_id_field_
name
The name of the HTML field in the current UI that you would like to write the resulting document identifier to. The resulting document identifier is determined by the document the user selects from the document management Search function. The document identifier is a concatenation of the following values:

DM:*<node_id>*:*<document_id>*:*<version>*

<nodeid> is the node ID assigned to the document management system node as defined in the Document Management Nodes Web page.

<documentid> is the document ID of the document, as assigned by the document management system where the document resides.

<version> is the version of the document. If a version is not specified, the latest version is assumed.

document_name_field_
name
The name of the HTML field in the current UI that you would like to write the resulting document name to.

Glossary

Access Level

A numeric value ranging from 0 to 1000. Every workflow user operates at a specific access level. The access level defines whether the user can modify certain workflow data. You can only modify data that is protected at a level equal to or higher than your access level.

Activity

A unit of work performed during a business process.

Activity Attribute

A parameter that has been externalized for a function activity that controls how the function activity operates. You define an activity attribute by displaying the activity's Attributes properties page in the Activities window. You assign a value to an activity attribute by displaying the activity node's Attribute Values properties page in the Process window.

Agent

A named point of communication within a system.

Agent Listener

A type of service component that processes event messages on inbound agents.

Attribute

See Activity Attribute, Item Type Attribute, or Message Attribute.

Background Engines

A supplemental Workflow Engine that processes deferred or timed out activities or stuck processes.

Business Event

See Event.

Cost

A relative value that you can assign to a function or notification activity to inform the Workflow Engine how much processing is required to complete the activity. Assign a higher cost to longer running, complex activities. The Workflow Engine can be set to operate with a threshold cost. Any activity with a cost above the Workflow Engine threshold cost gets set to 'DEFERRED' and is not processed. A background engine can be set up to poll for and process deferred activities.

Concurrent Manager

An Oracle Applications component that manages the queuing of requests and the operation of concurrent programs.

Concurrent Process

An instance of running a non-interactive, data-dependent function, simultaneously with online operations. Each time you submit a request, a concurrent manager processes your request, starts a concurrent process, and runs a concurrent program.

Concurrent Program

A concurrent program is an executable file that performs a specific task, such as posting a journal entry or generating a report.

Concurrent Queue

A list of concurrent requests awaiting completion by a concurrent manager. Each concurrent manager has a queue of requests waiting in line to be run. If your system administrator sets up your Oracle Application to have simultaneous queuing, your request can wait to run in more than one queue.

Directory Services

A mapping of Oracle Workflow users and roles to a site's directory repository.

Event

An occurrence in an internet or intranet application or program that might be significant to other objects in a system or to external agents.

Event Activity

A business event modelled as an activity so that it can be included in a workflow process.

Event Data

A set of additional details describing an event. The event data can be structured as an XML document. Together, the event name, event key, and event data fully communicate what occurred in the event.

Event Key

A string that uniquely identifies an instance of an event. Together, the event name, event key, and event data fully communicate what occurred in the event.

Event Message

A standard Workflow structure for communicating business events, defined by the datatype `WF_EVENT_T`. The event message contains the event data as well as several header properties, including the event name, event key, addressing attributes, and error information.

Event Subscription

A registration indicating that a particular event is significant to a system and specifying the processing to perform when the triggering event occurs. Subscription processing can include calling custom code, sending the event message to a workflow process, or sending the event message to an agent.

External Functions

Programs that are executed outside of the Oracle Database.

External Java Functions

Java programs that are executed outside of the Oracle Database by the Java Function Activity Agent.

Function

A PL/SQL stored procedure that can define business rules, perform automated tasks within an application, or retrieve application information. The stored procedure accepts standard arguments and returns a completion result.

Function Activity

An automated unit of work that is defined by a PL/SQL stored procedure.

Generic Service Component Framework

A facility that helps to simplify and automate the management of background Java services.

Item

A specific process, document, or transaction that is managed by a workflow process.

Item Attribute

See Item Type Attribute.

Item Type

A grouping of all items of a particular category that share the same set of item attributes. Item type is also used as a high level grouping for processes.

Item Type Attribute

A feature associated with a particular item type, also known as an item attribute. An item type attribute is defined as a variable whose value can be looked up and set by the application that maintains the item. An item type attribute and its value are available to all activities in a process.

Lookup Code

An internal name of a value defined in a lookup type.

Lookup Type

A predefined list of values. Each value in a lookup type has an internal and a display name.

Message

The information that is sent by a notification activity. A message must be defined before it can be associated with a notification activity. A message contains a subject, a priority, a body, and possibly one or more message attributes.

Message Attribute

A variable that you define for a particular message to either provide information or prompt for a response when the message is sent in a notification. You can use a predefine item type attribute as a message attribute. Defined as a 'Send' source, a message attribute

gets replaced with a runtime value when the message is sent. Defined as a 'Respond' source, a message attribute prompts a user for a response when the message is sent.

Node

An instance of an activity in a process diagram as shown in the Process window.

Notification

An instance of a message delivered to a user.

Notification Activity

A unit of work that requires human intervention. A notification activity sends a message to a user containing the information necessary to complete the work.

Notification Mailer

A type of service component that sends e-mail notifications to users through a mail application, and processes e-mail responses.

Notification Worklist

A Web page that you can access to query and respond to workflow notifications.

Performer

A user or role assigned to perform a human activity (notification). Notification activities that are included in a process must be assigned to a performer.

Process

A set of activities that need to be performed to accomplish a business goal.

Process Definition

A workflow process as defined in Oracle Workflow Builder, which can be saved as a flat file or in a database.

Process Activity

A process modelled as an activity so that it can be referenced by other processes.

Protection Level

A numeric value ranging from 0 to 1000 that represents who the data is protected from for modification. When workflow data is defined, it can either be set to customizable (1000), meaning anyone can modify it, or it can be assigned a protection level that is equal to the access level of the user defining the data. In the latter case, only users operating at an access level equal to or lower than the data's protection level can modify the data.

Result Code

The internal name of a result value, as defined by the result type.

Result Type

The name of the lookup type that contains an activity's possible result values.

Result Value

The value returned by a completed activity.

Role

One or more users grouped by a common responsibility or position.

Service Component Container

An instance of a service or servlet that manages the running of the individual service components that belong to it. The container monitors the status of its components and handles control events for itself and for its components.

Service Component

An instance of a Java program which has been defined according to the Generic Service Component Framework standards so that it can be managed through this framework.

Subscription

See Event Subscription.

System

A logically isolated software environment such as a host machine or database instance.

Timeout

The amount of time during which a notification activity must be performed before the Workflow Engine transitions to an error process or an alternate activity if one is defined.

Transition

The relationship that defines the completion of one activity and the activation of another activity within a process. In a process diagram, the arrow drawn between two activities represents a transition.

Workflow Definitions Loader

A concurrent program that lets you upload and download workflow definitions between a flat file and a database.

Workflow Engine

The Oracle Workflow component that implements a workflow process definition. The Workflow Engine manages the state of all activities for an item, automatically executes functions and sends notifications, maintains a history of completed activities, and detects error conditions and starts error processes. The Workflow Engine is implemented in server PL/SQL and activated when a call to an engine API is made.

Index

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W